Universidad
Carlos III de Madrid
www.uc3m.es

TRABAJO FIN DE GRADO

Integración de encoders absolutos en el control distribuido de Manfred 3

Autora: Gloria Sepúlveda García

Titulación: Grado en Ingeniería en Tecnologías Industriales

Tutor: Doctor Antonio Flores Caballero

Director: Profesor Luis Enrique Moreno Lorente

Fecha: Junio 2015

ÍNDICE

1. INTRODUCCIÓN

-Control de sistemas mecatrónicos

A lo largo de la historia se han desarrollado diferentes tecnologías con el fin de solucionar las trabas a las que se enfrenta la humanidad. En principio, las iniciativas se basaban en la experimentación; ensayo y error. Sin embargo, también encontramos reflexiones cuidadosas que aplican los conocimientos y técnicas para facilitar la realización de procesos de una forma eficaz [1]. De este concepto surge la ciencia y su relación innegable con la tecnología. Podemos destacar los avances que surgen en la Revolución Industrial como gran cambio en los sistemas productivos. La aparición de nuevas máquinas proporcionaba la fuerza, la destreza y agilidad a los procesos que tradicionalmente llevaba a cabo un operario.

Cada vez que un problema afectaba a cualquier componente, había que recurrir por separado a profesionales especialistas en ese área. Era muy difícil ponerlos de acuerdo sobre la solución del inconveniente, ya que cada profesional manejaba terminología y conceptos diferentes. En este punto, se fundamenta la llamada ciencia mecatrónica. Es un acrónimo de mecánica y electrónica que refleja la sinergia entre distintas tecnologías (Ver *Ilustración 1*).

Busca lograr convertirse en una ingeniería capaz de aportar lo mejor de cada área para crear productos inteligentes, con mejores cualidades respecto a las demás, capaces de procesar paralelamente diversas informaciones para optimizar el funcionamiento, mejorar la productividad y el desempeño. La integración interdisciplinaria es el recurso más importante que posee esta nueva tecnología, y la que le da el valor agregado respecto a otras. Permite contextualizar las etapas de desarrollo de proyectos complejos.

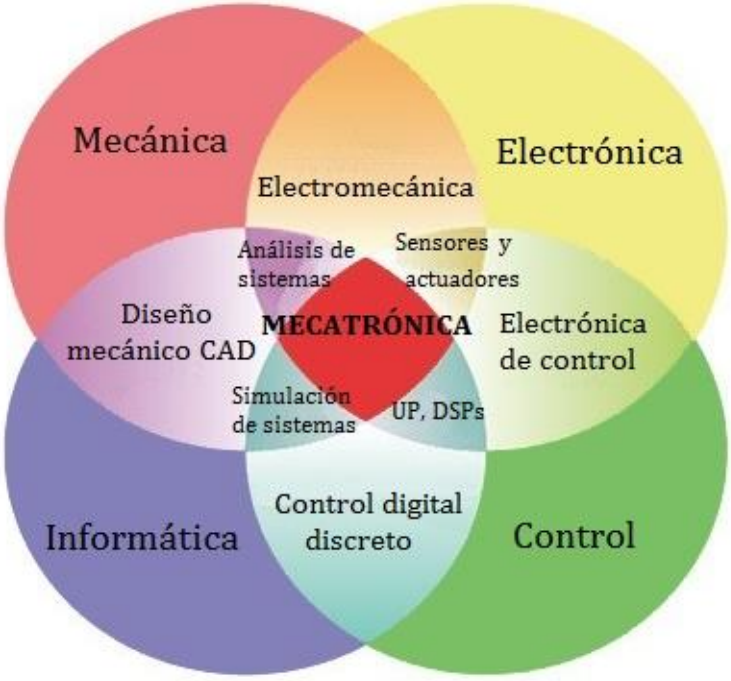


Ilustración 1: Mecatrónica como integración de disciplinas científicas.

La mecatrónica ha evolucionado en la medida que se han podido integrar los avances logrados en diversas disciplinas. A pesar de no poder hablar de fechas exactas, el crecimiento ha sido evidente. Históricamente el proceso se divide en tres etapas básicas. Desde finales de 1978 hasta el comienzo de 1980 fue el periodo en el cual se introdujo el término en el medio industrial, y se buscó su aceptación. Cada una de las ingenierías que ahora abarca la mecatrónica se desarrollaba independientemente. Posteriormente, en la década de 1980 se consolida la interdisciplinariedad de la nueva ciencia y se acuña el término a partir de la experiencia de un fabricante de robots en Japón [2].

Finalmente hacia los 90 se inicia el desarrollo de la inteligencia computacional y los sistemas de información. Una característica importante de esta última etapa es la miniaturización de los componentes en forma de micro procesadores y micro sensores, integrados en sistemas micro electromecánicos. Actualmente, la era digital dirige el rumbo de la mecatrónica, aplicada al desarrollo de software y hardware para computadores, máquinas y sistemas inteligentes y automatización industrial.

La Robótica, tanto industrial como de servicio, se puede considerar un caso específico de sistema mecatrónico aplicado al diseño, construcción y explotación de dispositivos genéricos orientados a potenciar el trabajo humano, y en algunos casos sustituirlo para evitar riesgos laborales en tareas repetitivas de manufactura y automatización de procesos industriales. Podemos destacar como ejemplo práctico el robot Manfred 2 que se encuentra en el RoboticsLab de la UC3M sobre el que partiremos para plantear nuestro problema a resolver en este proyecto (Ver *Ilustración 2*) [3].

Ilustración 2: Robot Manfred 2.

Se pueden apreciar tres partes esenciales en cualquier sistema mecatrónico que son [4]: sensores, control y actuadores. Los sensores o transductores de entrada: aportan información del exterior al sistema. El mundo físico se encuentra dominado por señales analógicas, no eléctricas, así que el trabajo del sensor es transformar estas señales para que la etapa de control pueda procesarlas. Control: es el cerebro de nuestro sistema. Aquí se analizan los datos y se toman decisiones. Actuadores o transductores de salida: despliegan la información obtenida. La señal eléctrica es convertida en otro tipo.

En nuestro caso, los sensores se encuentran en las articulaciones del brazo robótico. Son encoders; nos dan datos sobre las posiciones del mismo. Externamente, se procesan en la torre de control. Los actuadores son los motores, que proporcionan potencia. Estos elementos son útiles conjuntamente como trabajo integrado. Integrar significa [5] "Agregar una cosa o elemento para completar un todo" en esta definición nos apoyaremos para entender la necesidad de este concepto en un sistema industrial automatizado. Un proceso se divide en distintos niveles. Existe una pirámide de automatización definida por la ISO donde se clasifican las etapas de estos sistemas (Ver *Ilustración 3*).

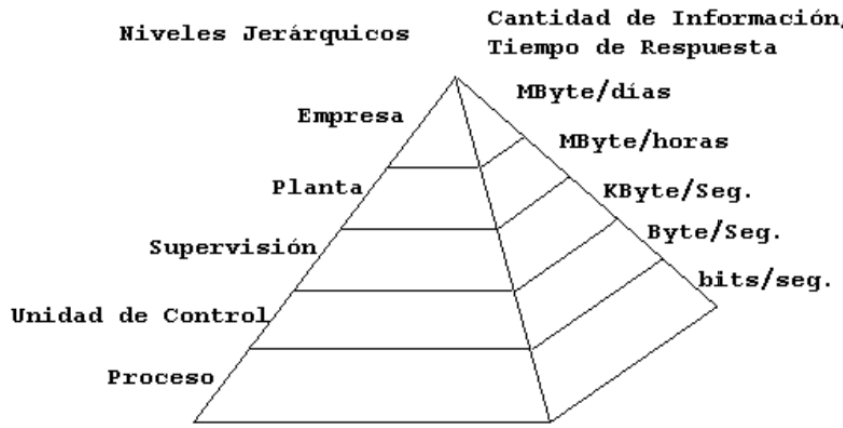


Ilustración 3: Modelo ISO de integración piramidal para automatización. Fuente: [6]

Los componentes de una solución de control, sean estos tangibles (hardware) o intangibles (software), están estrechamente vinculados unos con otros para lograr un fin común. El poder de complementarse para conseguir un resultado supone que se puedan asumir proyectos complejos. En un principio, esta integración resultaba engorrosa, dado que la gran mayoría de los sistemas ofrecidos requerían una especialización avanzada.

Hace ya bastante tiempo, se ofrecen tecnologías de sistemas de control abiertos. Las tecnologías menos restrictivas implican que el sistema de control completo debe ser diseñado por un especialista. Exigirá más trabajo, costará menos en materiales, no se dependerá tanto de las herramientas de terceras compañías y a la larga resultará más barato y eficiente, así como será más sencillo integrar nuevos elementos.

Actualmente, se ha avanzado mucho, pero aún continúa habiendo dificultades importantes; por ejemplo con el uso de códigos de programación propietarios o muy

restrictivos. Este problema es muy frecuente en la industria, ya que las empresas limitan el diseño de sus productos para su propio beneficio. Con el robot Manfred 2 plasmamos el impedimento que esto supone a la hora de desarrollar una aplicación determinada. Nos encontramos muchas limitaciones al programar ya que estamos condicionados a los requerimientos del fabricante. De forma visual, es como si la pieza de un puzle no pudiese encajar en el conjunto (Ver *Ilustración 4*).

Ilustración 4: Dificultades en la integración de tecnologías.

La integración se extiende a múltiples campos [7]. Ha sido muy importante para la creación de los llamados aparatos inteligentes, la masificación de las redes de comunicación industrial, la informatización en la gestión de los procesos industriales. Trayendo con esto el mejor control de plantas industriales y como resultados directos mejoras en la productividad y en la calidad de productos. Ha abierto horizontes para nuevos proyectos que sin las tecnologías actuales nunca hubiesen sido rentables. Aumenta la seguridad en el control de variables críticas las cuales en ciertos casos conlleva beneficios indirectos; como son la disminución de emisiones contaminantes.

Además, es un pilar relevante en el sector de las tecnologías de la información y comunicación (TICs). La interoperabilidad a raíz de la existencia de diversas plataformas tecnológicas en el ámbito de las tecnologías de la información y la comunicación, que en la mayoría de los casos son de carácter propietario y la generación de contenidos. Un campo que está adquiriendo un interés y un protagonismo creciente en todos los sectores. Conforma junto con la conectividad, la convergencia y la seguridad una definición completa de las TICs.

La combinación de los últimos avances en la tecnología de las comunicaciones y su integración a los sistemas mecatrónicos están dando paso al surgimiento de los denominados sistemas ciber-físicos, posibilitando formas de interacción remota y de tele-presencia que abren nuevos paradigmas en la explotación de máquinas inteligentes al no requerir la presencia física del operador como, por ejemplo, los drones en aplicaciones de defensa, prospección y otras.

-Caso particular

Como ya hemos comentado en el epígrafe anterior, uno de los problemas que podemos encontrarnos en un robot industrial es la limitación de diseño. En concreto, este proyecto surge como respuesta a los obstáculos presentes a la hora de trabajar con Manfred 2. Se decidió diseñar un nuevo sistema que ofreciese más libertad. Es el llamado Manfred 3. Para que sea posible su uso hay que tomar una serie de medidas para adaptarlo a nuestra aplicación. Ya que hasta integrar todos sus componentes, como sucede con su predecesor en la imagen 2, es indispensable trabajos previos al montaje del mismo.

Uno de los inconvenientes que hallamos es el reducido ancho de banda puesto que son aparatos pensados para incluirse en una línea de producción. Los tiempos de los procesos son lentos porque dependen de varias máquinas. El trabajo con el nuevo robot, consiste en hacerlo más versátil y fiable, desde el punto de vista de su programación y sencillez de integración. Nuestro objetivo es que sea útil en trabajos que requieren cambios inmediatos en su comportamiento, como cancelar un movimiento en ejecución y sustituirlo por otro con toda facilidad. De por sí la electrónica industrial no nos permite esta aplicación. Para compatibilizar la realidad disponible con los conceptos teóricos usamos la integración de sistemas.

El objeto de estudio, Manfred 3, se compone de siete eslabones (Ver *Ilustración 5*). Las tres primeras forman el hombro, la cuarta junto con la quinta componen el codo y las dos últimas son la muñeca. Es en el extremo donde posteriormente se colocará una herramienta. Cada articulación incluye un actuador (motor) y un sensor (encoder). Ambos confluyen en el driver a través del cual podemos extraer información para conocer el estado del elemento donde se encuentran. Este TFG tiene como objetivo la integración de los sistemas de sensorización basados en encoders absolutos en la electrónica de control.

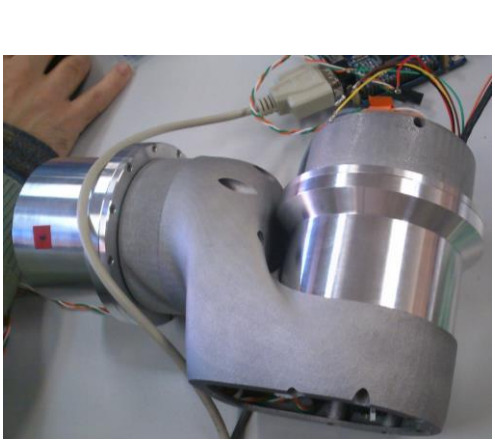


Ilustración 5: Articulaciones del robot Manfred 3.

La característica principal que hace útil un robot es el movimiento. Esto implica la definición de una trayectoria. Para ello se ha divido el proyecto en dos etapas. La primera, que es este caso, se encarga de recopilar los datos de los sensores (Ver *Ilustración 6*). La segunda, posteriormente los procesará correctamente para crear los recorridos de actuación.

Concretamente nuestra meta es ser capaces de interceptar correctamente la información de los sensores. Como se puede observar en la imagen posterior, cada dispositivo generará un tren de pulsos que corresponden a los bits de la palabra que indica la posición de cada articulación. Estas señales presentan una frecuencia de 500kHz. Además, la tasa de ejecución con que se reciben los impulsos es de 1000 veces/segundo.

También es importante para el posterior desarrollo del programa destacar el distanciamiento en tiempo de las lecturas según el eslabón al que correspondan. Es una cantidad que no podemos medir, pero tenemos en cuenta para la correcta reconstrucción de los datos.

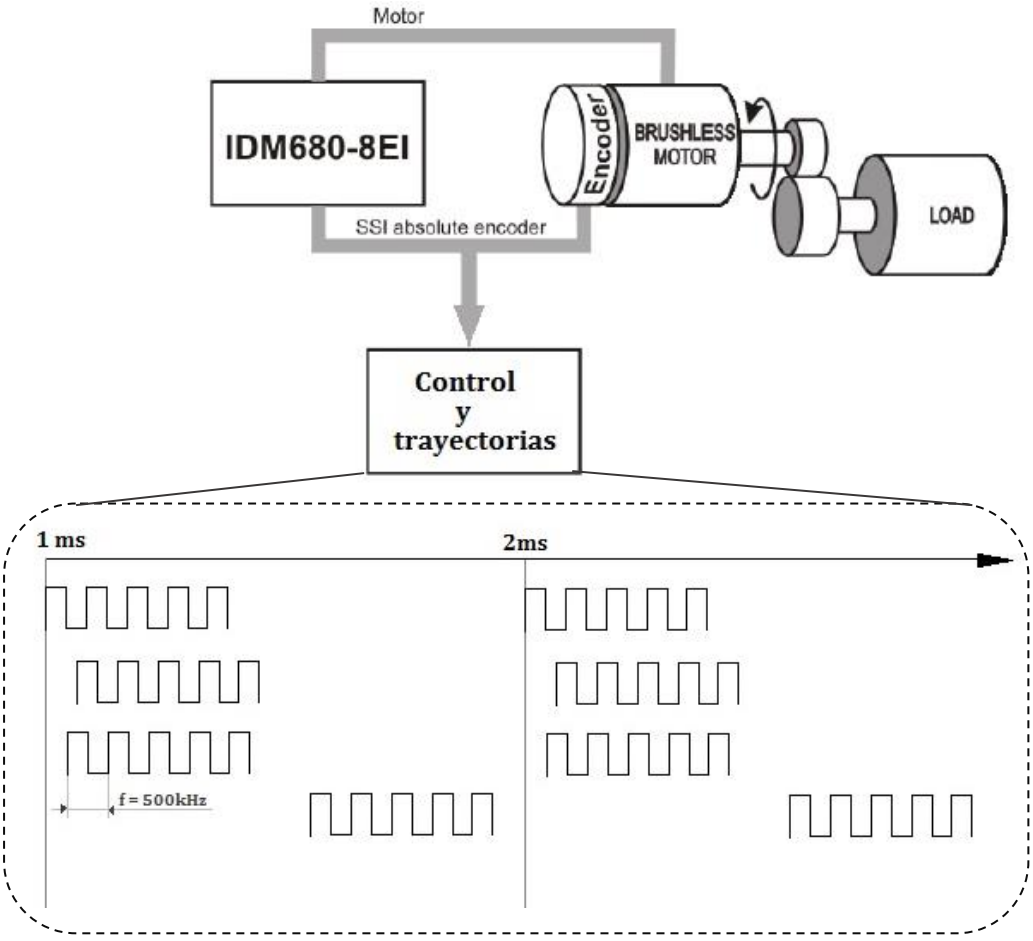


Ilustración 6: Diagrama de bloques del proyecto y detalle del sistema de control.

El entorno de trabajo se basa en la aplicación del concepto ARCP (del inglés Advanced Rapid Control Prototyping, en castellano Sistema Avanzado de Prototipado Rápido para Control) expuesto por el doctor Antonio Flores Caballero en su tesis doctoral [12]. Se trata de un nuevo enfoque dado al método de prototipado rápido de control.

Define: "el concepto de Prototipado Rápido para Control (RCP, del inglés "*Rapid Control Prototyping*") hace referencia a todas las técnicas software y hardware necesarias para acortar los tiempos de desarrollo y puesta en marcha de sistemas de control, haciendo uso de un alto nivel de abstracción en la programación." En nuestro caso se le añade la palabra *Advanced* para referirnos a los cambios que implica respecto al modelo tradicional.

Originalmente el uso de sistemas RCP se ha entendido como un paso intermedio entre el planteamiento teórico del controlador en el laboratorio y su implementación final. Sin embargo, con la solución adoptada por el UC3M RoboticsLab se cubren ambas etapas como si fueran una sola, con el controlador definitivo, eliminando de la fase de desarrollo la necesidad de una gran cantidad de personal cualificado, tiempo y recursos económicos. (Ver *Ilustración 7*).

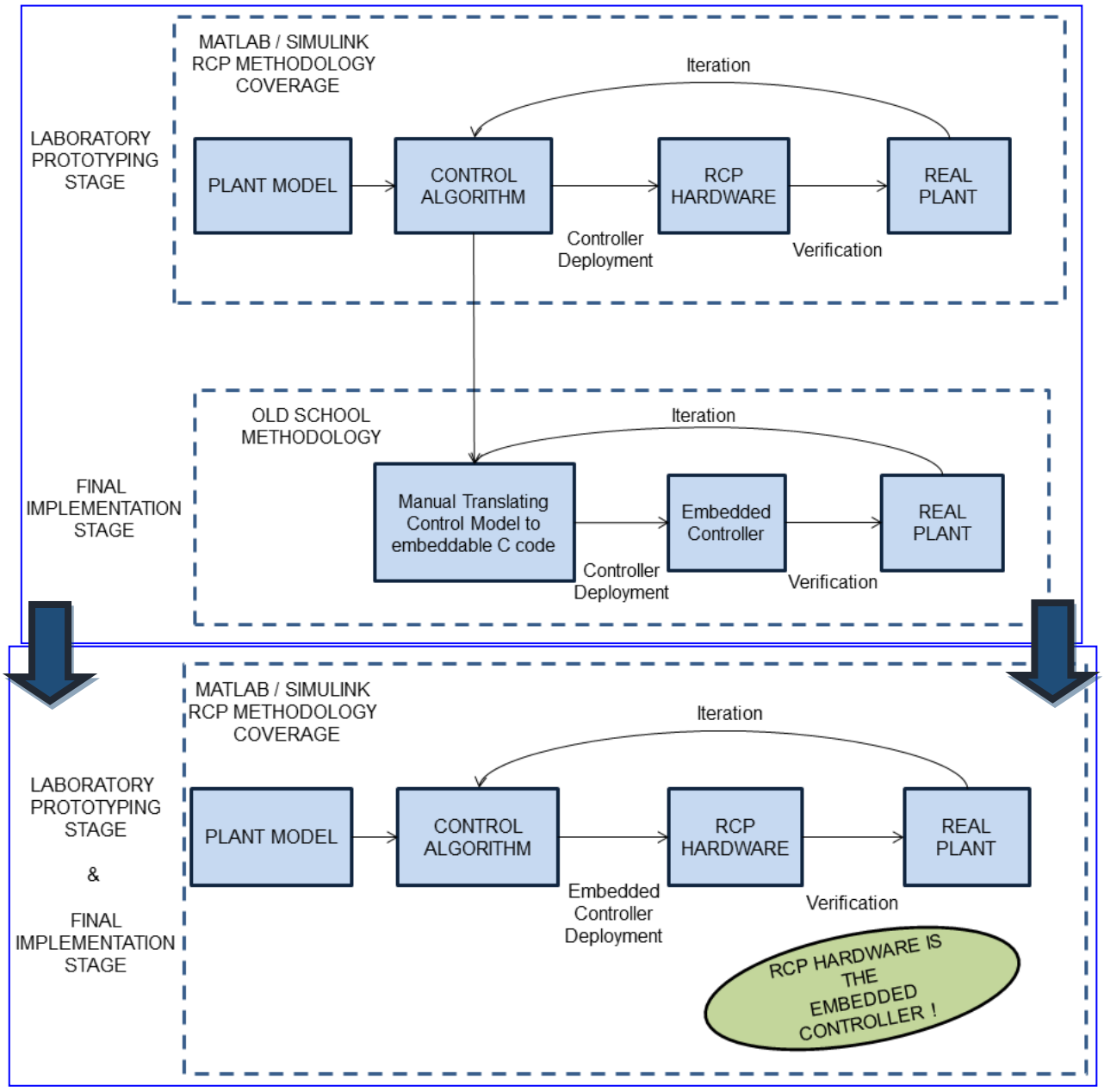


Ilustración 7: Comparación del sistema RCP comercial (arriba) frente al ARCP (abajo). Fuente: [12].

Se obtienen unos resultados mucho más satisfactorios pues la implementación final cumple ahora al 100% el modelo de control. En el momento en que el sistema funciona en el laboratorio, queda listo para su producción masiva, sin necesidad de recurrir a otro hardware de control al que traspasar los resultados reflejados en la experimentación.

Para cumplir este objetivo hace falta disponer de un controlador digital disponible en el mercado de forma industrializada, producido en serie a bajo coste y orientado a la implementación final, cuyas características computacionales y facilidad de manejo lo hagan apto también para su uso como sistema RCP de laboratorio.

En general, las estructuras empotradas de los microcontroladores cumplen con dichos requisitos. No suelen administrarse mediante sistemas operativos al uso, lo cual nos permite conocer en todo momento su estado y evitaremos imprevistos. Proporcionan procesamiento extremadamente barato, ya que pueden aprovechar las economías de escala. MCUs son programables en software, por lo que un fabricante de chips puede diseñar un único tipo que satisfaga las necesidades de muchos clientes. Esto reduce el coste por chip por amortizarlos durante muchos millones de unidades.

El precio de un circuito integrado (tal como un microcontrolador o un microprocesador) depende de dos factores [8]: ingeniería no recurrente y recurrente. El primero incluye el pago de los ingenieros para diseñar el circuito integrado (IC) y verificar a través de la simulación y la creación de un prototipo que funcione correctamente. En el segundo se incurre al hacer cada IC adicional, e incluye materias primas, procesamiento, pruebas y embalaje. El factor recurrente que es determinante es el área. Se aprovechan las obleas de silicio para hacer el mayor número posible de µC.

Además, las características propias de un microcontrolador complementan las funciones de un robot. El ambiente de trabajo de estas máquinas se rige por la llamada regla de las 4 Ds: labores repetitivas (*dull work*), sucias (*dirty work*), peligrosas (*dangerous work*) y difíciles (*difficult work*). Son un tipo de automatización flexible que deben tomar decisiones de forma autónoma. Los sistemas embebidos suelen ser de tamaño y peso reducidos y operan de forma robusta y fiable para que pueda trabajar de forma independiente.

La consecuencia de ello es reducir la necesidad del mantenimiento y facilitar la movilidad del robot. Conseguimos también evitar interferencias en el intercambio de información porque se reducen al máximo los cables. Su estructura es en tiempo real, es decir, que tienen duración de ejecución fija. El tiempo transcurrido entre la detección de eventos y la respuesta es crítico, ya que las respuestas tardías conllevan errores.

Por último, no se puede dejar de lado la seguridad en el análisis. Es necesario destacar una consecuencia directa del cambio de finalidad del robot. Originalmente su campo de trabajo no estaba pensado para ser compartido con personas. Al menos no de forma frecuente, solo en caso de reparación o alguna situación excepcional.

Ahora, en nuestra aplicación, tenemos que contar con la presencia de personas dentro del rango de movimientos del mismo. Existen normas internacionales que se centran principalmente en evitar el contacto robot-operario. Se distinguen diferentes áreas según el riesgo existente (Ver *Ilustración 8*). Se delimita con protecciones. La línea divisoria que acota la zona roja no debe ser sobrepasada en caso de fallos previsibles.

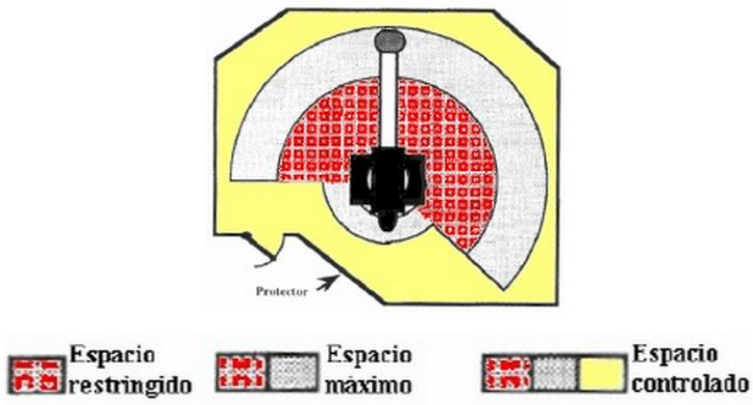


Ilustración 8: Clasificación del espacio de trabajo de un robot industrial. Fuente: [9].

Nuestro objetivo específico consiste en diseñar el control desde un punto de vista interno. Por ello, las protecciones van a ser diferentes a los aspectos externos mencionados anteriormente. La IEC (*International Electrotechnical Commision*) [10] define la seguridad funcional como "la parte de la seguridad general que depende de que un sistema o equipo funcione correctamente en respuesta a sus entradas. Es la detección de una condición potencialmente peligrosa que resulta en la activación de un dispositivo o mecanismo." Para evitarlos, establece una serie de normas de evaluación para todas las tecnologías eléctricas, electrónicas y relacionadas. En concreto nos interesa la IEC 61508. Las medidas pueden ser protectoras o correctivas.

IEC 61508 especifica 4 niveles de rendimiento de seguridad [11]. Estos son los llamados niveles de integridad de seguridad: SIL (*Safety Integrity Level*). Se numeran desde el más bajo que es el 1 hasta el más alto que es el 4. Estos requisitos son más rigurosos en los niveles más altos a fin de lograr la menor probabilidad requerida de fallo peligroso [12]:

- SIL1: Sin necesidad de Tiempo-Real. Programas de tamaño reducido. Fallos del sistema no implican daños a personas, financieros, a circuitos ni a elementos mecánicos. No hay elementos mecánicos. No hay bucles de control. Ejemplo: Programación de entrenamiento, prueba y error.

- SIL2: Algunas partes del programa requieren Tiempo-Real. Fallos del sistema pueden dañar el interés de grupos de personas, provocan pequeños riesgos financieros, provocan pequeños daños a elementos mecánicos. No hay bucles de control o no son relevantes. Estándar Recomendado: ISO9001. Ejemplo: Impresora 3D doméstica.

- SIL3: Fuerte contenido de Tiempo-Real. Fallos del sistema pueden dañar físicamente a grupos de personas, provocar riesgos financieros, provocar daños a elementos mecánicos, provocar la destrucción de electrónica de potencia. Bucles de control sencillos basados en PID. Estándar Recomendado: TickIT, ISO9001, IEC 61508. Ejemplo: Dispositivos médicos no críticos en seguridad, dispositivos industriales no críticos en seguridad.

- SIL4: Completamente basados en Tiempo-Real. Se requieren tiempos de puesta en marcha reducidos, del orden de milisegundos. Se requiere tolerancia/recuperación frente a fallos de funcionamiento. Pueden dañar de forma grave y físicamente a individuos, provocar grandes riesgos financieros, daños a elementos mecánicos, la destrucción de electrónica de potencia, la pérdida de dispositivos que no pueden recuperarse para ser reparados. Bucles de control sencillos y complejos/adaptativos.

Estándar Recomendado: TickIT, ISO9001, DO178/B/C, IEC 60880, ISO 26262. Ejemplo: Dispositivos médicos críticos en seguridad, dispositivos industriales críticos en seguridad, control del motor y frenos y transmisión automática de un automóvil, control de sistemas de transporte público, control de sistemas nucleares.

Atendiendo a los criterios del proyecto y a la clasificación anterior, nuestro sistema ARCP para el control de los encoders deberá tener un nivel de seguridad cuatro.

2. DESARROLLO DEL TRABAJO

Como se trata de un problema práctico, el método de trabajo hasta alcanzar el propósito final es iterativo. Consiste en definir un primer planteamiento para afrontar el tema. Un objetivo específico es realizar el control de forma unificada utilizando un solo microcontrolador. Para que sea factible hay que administrar eficientemente los medios disponibles.

Desarrollamos el algoritmo consecuentemente y realizamos las pruebas de laboratorio oportunas para comprobar si obtenemos el resultado esperado. Por último, si no son válidos, procedemos a corregirlos. Teniendo esto en cuenta cabe destacar que la estructura de la memoria no se corresponde con el seguimiento real práctico. Hemos clasificado el progreso en tres capítulos generales.

Inicialmente, analizamos las características propias del material que vamos a usar. Clasificamos los recursos en físicos y soporte informático. Más adelante, concretamos el código utilizado para desarrollar nuestra aplicación. Seguidamente comentamos las mediciones y valores obtenidos en los ensayos de laboratorio. Para acabar, revisamos si hemos satisfecho nuestros propósitos. Además plantearemos cuál es el coste aproximado del proyecto.

-Recursos materiales

▫ Hardware

El componente principal de estudio son los sensores de posición del robot. Se trata de encoders. También conocido como codificador o decodificador, es un dispositivo, circuito, programa de software, un algoritmo o incluso hasta una persona cuyo objetivo es convertir información de un formato a otro con el propósito de estandarización, velocidad, confidencialidad, seguridad o incluso para comprimir archivos [13].

Son utilizados en una infinidad de campos e industrias que van desde máquinas de fax, electro-domésticos de consumo, hasta robótica, minería transporte, aeroespacial y más. De los que hablaremos aquí son para control de motores. Su función es la de convertir el movimiento mecánico (giros del eje) en pulsos digitales o análogos que pueden ser interpretados por un controlador de movimiento.

Para explicar cómo funciona destacamos que se compone básicamente de un disco conectado a un eje giratorio. Está hecho de vidrio o plástico y se encuentra "codificado" con unas partes transparentes y otras opacas que bloquean el paso de la luz emitida por la fuente de luz (típicamente emisores infrarrojos). En la mayoría de los casos, estas áreas bloqueadas (codificadas) están situadas en forma radial. (Ver *Ilustración 9*).

A medida que el eje rota, el infrarrojo emite luz que es recibida por el sensor óptico (o foto-transistor) generando los pulsos digitales a medida que la luz cruza el disco o es

bloqueada. Esto produce una secuencia que puede ser usada para controlar el radio de giro, la dirección del movimiento e incluso la velocidad.

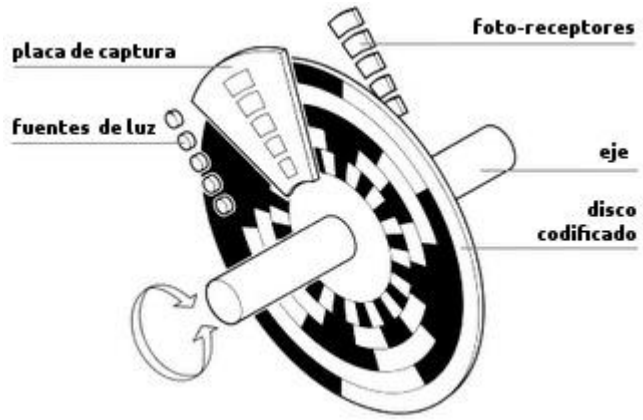


Ilustración 9: Componentes básicos de un encoder. Fuente: [13].

Existen básicamente dos tipos de encoder según su diseño básico y funcionalidad: encoder Incremental y encoder Absoluto. Adicionalmente existen otros tipos de encoders como por ejemplo el encoder óptico, lineal y el encoder de cuadratura.

El encoder óptico es el más usado (Ver *Ilustración 10*). Consta básicamente de tres partes: una fuente emisora de luz, un disco giratorio y un detector de luz conocido como "foto detector". El disco está montado sobre un eje giratorio y cuenta con secciones opacas y transparentes sobre la cara del disco. La luz que emite la fuente es recibida por el foto-detector o interrumpida por el patrón de secciones opacas produciendo señales de pulso. El código que se produce es leído por un dispositivo controlador que incluye un micro-procesador para determinar el ángulo exacto del eje.

Ilustración 10: Encoder óptico. Fuente: [42].

Un encoder lineal es un dispositivo o sensor que cuenta con una escala graduada para determinar su posición (Ver *Ilustración 11*). Los sensores leen la escala para después convertir su posición codificada en una señal digital que puede ser interpretada por un controlador de movimiento electrónico. Pueden ser absolutos o incrementales y existen diferentes tipos según la tecnología usada en su mecanismo, por ejemplo, tecnología óptica, magnética, inductiva o capacitiva. Se utiliza en aplicaciones de metrología, sistemas de movimiento y para controlar instrumentos de alta precisión en la fabricación de herramientas.

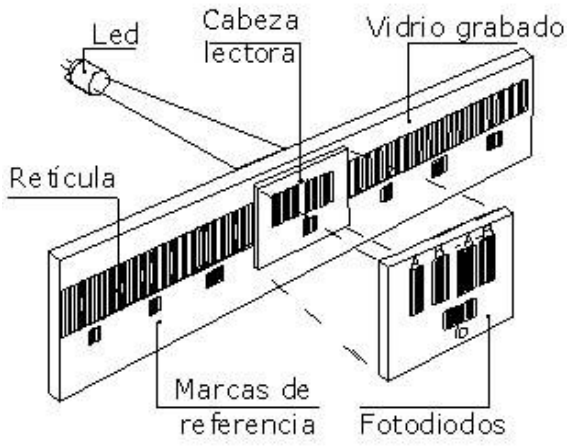


Ilustración 11: Encoder lineal. Fuente: [43].

Un encoder de cuadratura es un tipo de encoder rotativo incremental, el cual tiene la capacidad de indicar tanto la posición, como la dirección y la velocidad del movimiento. (Ver *Ilustración 12*). Los encoders de cuadratura se encuentran con mucha más frecuencia en muchos productos eléctricos de consumo y en una infinidad de aplicaciones comerciales.

La flexibilidad del encoder de cuadratura es su principal ventaja, ya que ofrecen una alta resolución, medición con precisión quirúrgica y pueden trabajar en un gran espectro de velocidades que van desde unas cuantas revoluciones por minuto hasta velocidades que van más allá de las 5000 RPM. Generalmente utiliza sensores ópticos o magnéticos, lo cual los convierte en dispositivos sencillos de usar y extremadamente duraderos.

Un encoder incremental, como su nombre lo indica, determina el ángulo de posición por medio de realizar cuentas incrementales. Esto quiere decir que provee una posición estratégica desde donde siempre comenzará la cuenta. La posición actual es incremental cuando es comparada con la última posición registrada por el sensor. Los encoders incrementales son de tipo óptico donde cada posición es completamente única.

Ilustración 12: Encoder incremental. Fuente: [41].

Un encoder absoluto se basa en la información proveída para determinar la posición absoluta en secuencia. (Ver *Ilustración 13*). Ofrece un código único para cada posición.

Se dividen en dos grupos: los encoders de un solo giro y los encoders absolutos de giro múltiple y su tamaño es pequeño para permitir una integración más simple. Los encoders absolutos son usados más comúnmente en motores eléctricos sin cepillos (*brushless motors*), en la medicina, la industria del transporte en especial en trenes, en la minería y otras.

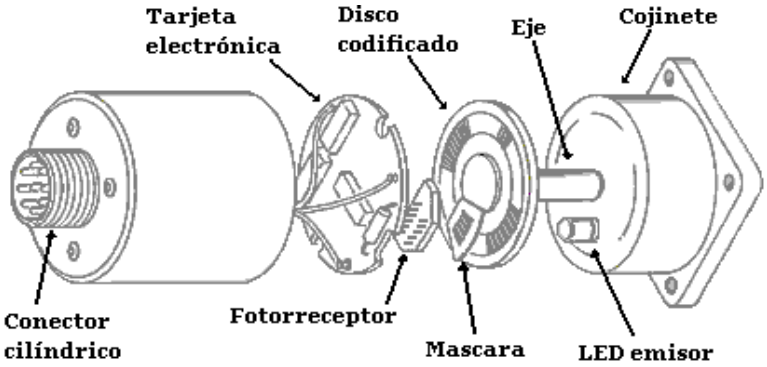


Ilustración 13: Encoder absoluto. Fuente: [44].

Los que tenemos para nuestro proyecto son encoders SSI de tipo absoluto diseñados por la empresa Netzer. Miden la posición de la articulación, en vez de la velocidad que sería su función si fuese relativo. SSI es un protocolo de comunicación serie para establecer la comunicación con los motores. En concreto se trata de RS-422. Un inconveniente de este encoder absoluto es que no puede leerse con una señal de reloj potencia de 2.

Hay dos tipos diferentes según la articulación donde estén situados: DS-58[20] para la muñeca y DS-70 para el hombro y el codo. Ambos tienen un funcionamiento parecido excepto en algún parámetro específico. El primero tiene una resolución angular de 18 bits, mientras que el segundo es de 19. Sin embargo, la velocidad máxima es superior en el extremo del brazo. Esta característica es consecuente con la aplicación del aparato, ya que es en esa articulación donde necesitamos mayor margen de trabajo porque es donde se acoplará la herramienta.

En lo que se refiere a comunicaciones entre dispositivos se distinguen varios planteamientos. Según el método de conexión separamos en paralelo y serie. Dependiendo del sentido del flujo de la información trabajaremos en modo *full duplex;* se permite la comunicación en dos direcciones al mismo tiempo o *half duplex*; en una sola dirección a la vez [8].

Si nos centramos en los enlaces, se pueden unir punto a punto. Se necesita una conexión separada para cada elemento. Una simplificación es permitir que todos los dispositivos compartan el mismo conjunto de líneas de datos, formando un bus de datos. En este caso tenemos un bus paralelo que utiliza una señal para cada una de las palabras (conjunto de ocho bits) de datos. Es preciso añadir líneas de control para indicar quién y qué debe activarse; por ejemplo si se lee o se escribe. Esto puede ser rápido y sencillo de implementar, pero requiere muchas señales, elevando tamaños de envases IC (debido a los pines) y tamaños de PCB (debido al número de trazas).

Para reducir los incovenientes, podemos reemplazar el bus de datos paralelo por un bus de datos en serie. Corresponden a este tipo de interfaz los estándares RS-232 y RS-422. Se realiza dicha conversión con registros de desplazamiento. La salida paralelo-serie se carga por primera vez con los datos de entrada utilizando los multiplexores para seleccionar los *input* paralelos y marcando esos datos en biestables tipo D. Las líneas de control del multiplexor son entonces volteadas de modo que los datos de entrada para cada *flip-flop* es la salida del *flip-flop* a la izquierda. Cada pulso de reloj posterior hará que los datos se desplacen a un bit hacia la derecha. (Ver *Ilustración 14*).

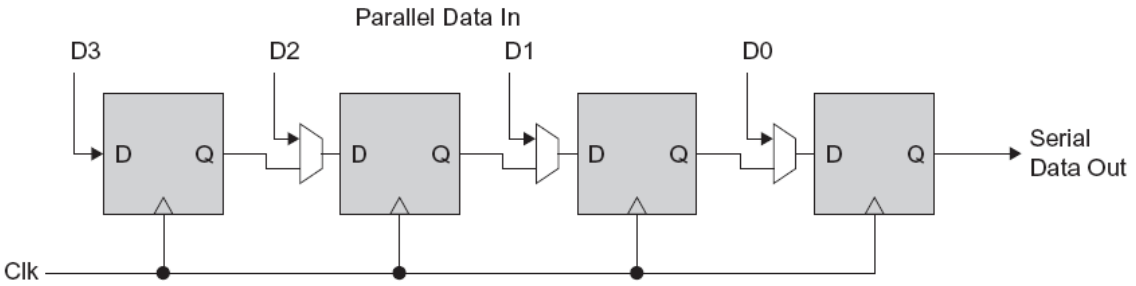


Ilustración 14: Implementación de la conversión paralelo-serie. Fuente: [8].

Cada periférico y MCU tiene registros de desplazamiento internos para llevar a cabo la conversión de ida y vuelta entre los formatos paralelo y en serie. Este es el enfoque utilizado para las comunicaciones de CSI y SPI. Hay que considerar que conntamos con líneas de señales separadas para transmitir y recibir datos (*data out* - *data in*). Esto queire decir que se usa *full duplex*.

Si la información de recepción y transmisión se envían en la misma línea (*half duplex*), reducimos aún más el número de señales necesarias. Nuestra comunicación de datos ahora usa un paquete con los datos y la información de direccionamiento. Los protocolos IIC utilizan esta estrategia con aplicaciones que tengan como requisito un limitado número de pines.

Por último, si eliminamos la línea de reloj explícita tenemos una comunicación asíncrona en serie. Este es el planteamiento que siguen los protocolos UART e integrados. Este es el comportamiento de nuestro bloque USB [14]. En este modelo cabe entender que ambos equipos poseen relojes funcionando a la misma frecuencia.

Para realizar la transmisión, se prepara un grupo de bits encabezados por una referencia de temporización que indica cuándo iniciar la toma de muestras con un bit de inicio conocido como de arranque, un conjunto de 7 u 8 bits de datos, un bit de paridad (para control de errores), y uno o dos bits de parada. El primero de los bits enviados anuncia al receptor la llegada de los siguientes, y la recepción de los mismos es efectuada. El receptor conoce perfectamente cuántos bits le llegarán, y da por recibida la información cuando verifica la llegada de los bits de parada.

Se denomina transmisión asincrónica no porque no exista ningún tipo de sincronismo, sino porque el sincronismo no se halla en la señal misma, más bien son los equipos mismos los que poseen relojes o *clocks* que posibilitan la sincronización. La sincronía o asincronía siempre se comprende a partir de la señal, no de los equipos de transmisión o recepción.

Los motores son de rotación sin escobillas. La conexión de dichos actuadores se realiza a través de un driver y no de forma directa hacia los sensores. El modelo utilizado es un IDM 680-8EI can open. La transferencia de datos al exterior se hace con el protocolo de comunicación serie RS-232 a 115200 baudios. Su función en nuestro sistema es actuar como maestro. Dirige el proceso a través de un tren pulsos de reloj que serán recibidos por el esclavo (encoder) para capturarlos. Para facilitar el procesado de la información realizamos con un micro STRIVE X221595 la conversión serie a USB para conectarnos al ordenador (Ver *Ilustración 15*).

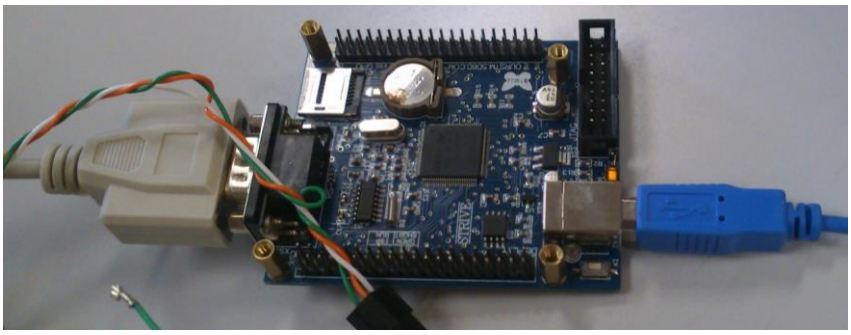


Ilustración 15: µC para conversión serie-USB.

De los manuales extraemos que las señales de la línea de datos del SSI se basan en una medida diferencial entre los polos +/-. Por tanto, para conectar los pines del encoder necesitamos tener una señal balanceada. Esta característica nos permite que el sistema sea más inmune al ruido y además evitemos el dilema de establecer un voltaje de referencia, ya que las medidas son diferenciales.

Para conseguir este comportamiento utilizamos el *transceiver* MAX3077E que trabaja con 3,3V. Un *transceiver* es un dispositivo que se encarga de convertir los niveles de tensión a los específicos de las normas utilizadas para la comunicación necesarios tanto a la entrada como a la salida de dicho sistema. Se puede traducir al español como transductor [15].

Por último, la clave del funcionamiento del trabajo se basa en un µC tipo STM32F4 del fabricante ST Microelectronics (Ginebra, Italia). (Ver *Ilustración 40*). Está compuesta por un procesador tipo ARM (*Advanced RISC Machines*) Cortex-M4. Las máquinas RISC (*Reduced Instruction Set Computer*) constituyen una "filosofía" de diseño de CPUs.

Consiste en instrucciones sencillas y optimizadas. Su ejecución se basa en registros de uso general. Presenta mayor frecuencia de reloj en comparación con máquinas CISC (*Complex Instruction Set Computer*). Este otro tipo utiliza registros y memoria para interpretar instrucciones complejas, lo cual complica la unidad de control. Para un mismo código se necesitan menos instrucciones. Por tanto, más periodo de reloj.

Conocer las características de la arquitectura del microcontrolador es importante para desarrollar el lenguaje ensamblador, ya que en cada máquina es diferente. No es posible hacer una comparativa que pueda declarar la superioridad de una u otra arquitectura. Esto se debe a varios factores. No existe una pareja de máquinas RISC y CISC directamente comparables. Los resultados pueden depender del conjunto de programas de prueba. Es difícil separar los efectos del hardware de la habilidad del compilador.

Sin embargo, lo que sí puede estudiarse son los aspectos ventajosos que implica la estructura RISC en nuestro µC [16]. Afectará a la sencillez de la unidad de control debido al hecho de poseer una longitud fija, formato de instrucción único (o pocos formatos de instrucción), pocos modos de direccionamiento, y operar básicamente con registros. De esta forma, las instrucciones pueden ejecutarse más rápidamente que en un CISC comparable.

Los programas RISC son más sensibles a interrupciones, ya que la comprobación es más frecuente que en otras arquitecturas con instrucciones complejas. La arquitectura RISC puede aplicar más eficazmente las técnicas de segmentación de instrucciones. Además, supone reducción del área. En un microprocesador CISC típico la unidad de control (CU) suele ocupar una gran parte del área en silicio (hasta el 50%), mientras que un RISC necesita un espacio mucho menor.

Desde otro punto de vista, podemos también relacionar las siglas con su utilidad: *Application Real time Microcontroller*. Implica una serie de ventajas [17]. Con una misma estructura hay opción a la adaptación según la implementación que sea necesaria. También contamos con gran cantidad de información libre y herramientas hardware y software para el desarrollo y depuración de tareas en este dispositivo. Está basado en arquitectura Harvard.

A continuación, señalamos las características de esta arquitectura, así como sus ventajas e inconvenientes respecto de la distribución Von Neumann. Podemos observar la principal diferencia en la *Ilustración 16*. Además la colocación es cronológica, ya que es la primera imagen la que corresponde al tipo de estructura utilizado en las primeras creaciones de sistemas de control basadas en microprocesador.

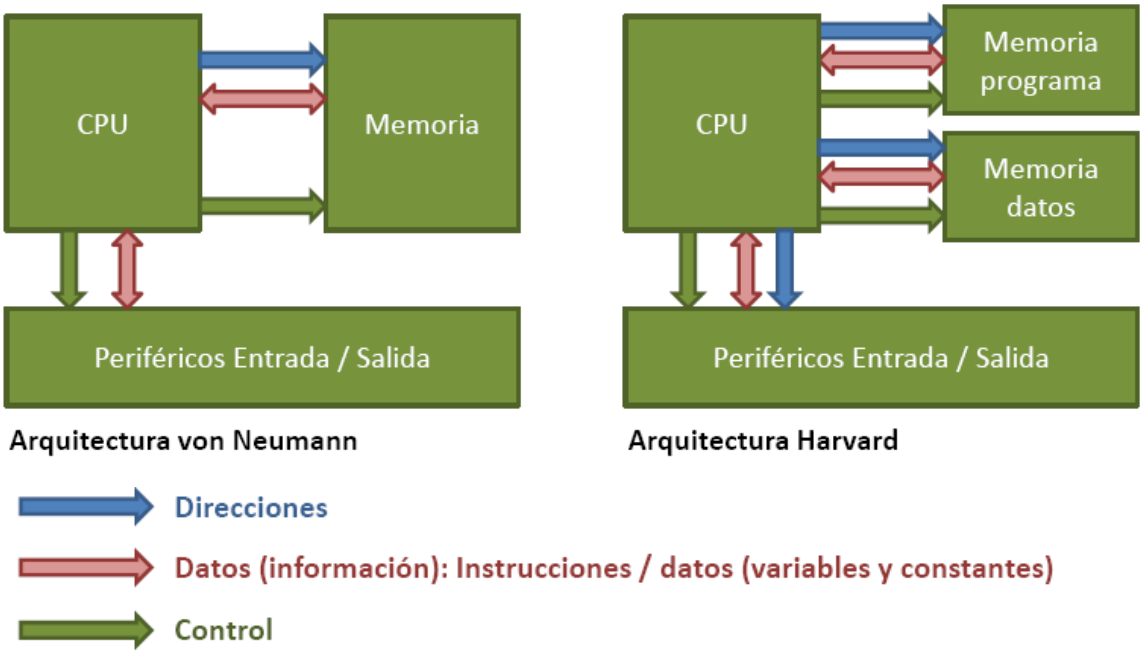


Ilustración 16: Posibles arquitecturas en un microprocesador. Von Neumann vs. Harvard. Fuente: [18].

La memoria es única, no se distingue el espacio para programa frente a los datos. Esta división si está físicamente clasificada en la segunda (tipo Harvard). El inconveniente de tener que compartir el bus de datos es que se limita el ancho de banda, ya que solo puede transmitirse una instrucción a la vez. Sin embargo, en la Harvard podemos aumentar el rendimiento del sistema porque se pueden simultanear.

Las características mencionadas anteriormente cumplen los objetivos de uso de un sistema embebido. Tenemos mayor fiabilidad. Podemos aprovechar el hardware de la placa debido a organización de memoria según necesitemos que sea constante o volátil. Por el contrario, también tenemos la aplicación limitada, porque si quisiésemos usar esta arquitectura para un procesador de carácter general, tendríamos impedimentos que se suplen con la implementación de Von Neumann. Por ejemplo, sería muy costoso hacer una ampliación de la memoria.

El microcontrolador es de 32 bits. Ofrece un montón de interfaces digitales y analógicas que no requieren complementos de hardware externo [19]. La potencia de cálculo del sistema funciona a 168-180 MHz con soporte de punto flotante a través de hardware. Es casi el doble de más del 50% de los dispositivos listos para la producción en el mercado de control industrial, que comúnmente requieren un MCU de 16-32 bits funcionando a 10-99 MHz sin soporte de hardware de punto flotante, y no más del 25% requieren velocidades de reloj más allá de 100 MHz. Estas características sumadas al mínimo consumo de energía lo hacen ideal para ser utilizado como sistema ARCP.

Manejamos esta placa desde dos enfoques. Por un lado, es donde cargaremos nuestro programa desde Matlab. Por otro, se emula el comportamiento del sensor real para poder trabajar de forma independiente al driver. Esta configuración se considera un dato fijo, de solo lectura, ya que es nuestra referencia. Escribimos en la primera que llamaremos sistema objetivo o *target*.

Dependiendo en qué fase de desarrollo nos encontremos, seleccionamos los medios que necesitemos. Por ejemplo, si el driver funcionase correctamente, nos ahorraríamos el integrado para polarizar las señales, porque por hardware ya tiene pines diferentes para que esas conexiones sean directas. (Ver *Ilustración 38*).

▫ Software

La empresa Technosoft que ha diseñado los drivers ofrece un programa específico llamado Easy Motion. Es una interfaz que nos facilita comprobar si el funcionamiento es correcto utilizando herramientas implementadas en dicho entorno. Para empezar podemos hacer un *test* y ver las lecturas de las posiciones que se registran.

Con fin de cumplir el objetivo de acortar tiempos usando ARCP es clave tener un alto nivel de abstracción en programación. [12] Se obtiene al utilizar lenguajes basados en gráficos, que permiten transferir las ideas del programador al computador de una forma más natural, obviando todos los aspectos de configuración a bajo nivel del sistema hardware, pues estos pasos se realizan automáticamente.

Conlleva ventajas incluso para los más expertos en programación textual. Un lenguaje de este tipo pone al alcance de un público multidisciplinar el hacer uso de un hardware de control real. También permite que dicho controlador se comporte exactamente tal y como se ha definido su comportamiento en el lenguaje gráfico que le ha servido de programación. Este último punto no sería posible si se tuviera que realizar una transcripción manual desde el lenguaje gráfico a un lenguaje textual, el programador introduciría modificaciones de forma involuntaria.

[20] Jack Little y Cleve Moler, los cofundadores de The MathWorks, vieron la necesidad entre los ingenieros y científicos de obtener entornos de computación más potentes y productivos más allá de lenguajes Fortran o C. En respuesta, combinaron sus conocimientos en matemáticas, ingeniería y ciencias de la computación para desarrollar MATLAB, un entorno de computación técnico de alto rendimiento. Combina funciones integrales de matemáticas y gráficos con un potente lenguaje de alto nivel.

Mediante el lenguaje de MATLAB, se pueden escribir programas y desarrollar algoritmos de manera más rápida que con los lenguajes tradicionales, ya que no es necesario realizar tareas administrativas de bajo nivel tales como declarar variables, especificar tipos de datos y asignar memoria. En muchos casos, el soporte para las operaciones de vectores y matrices elimina la necesidad de bucles For. Como resultado, una línea de código de MATLAB puede reemplazar varias líneas de código C o C++ (Ver *Ilustración 17*).

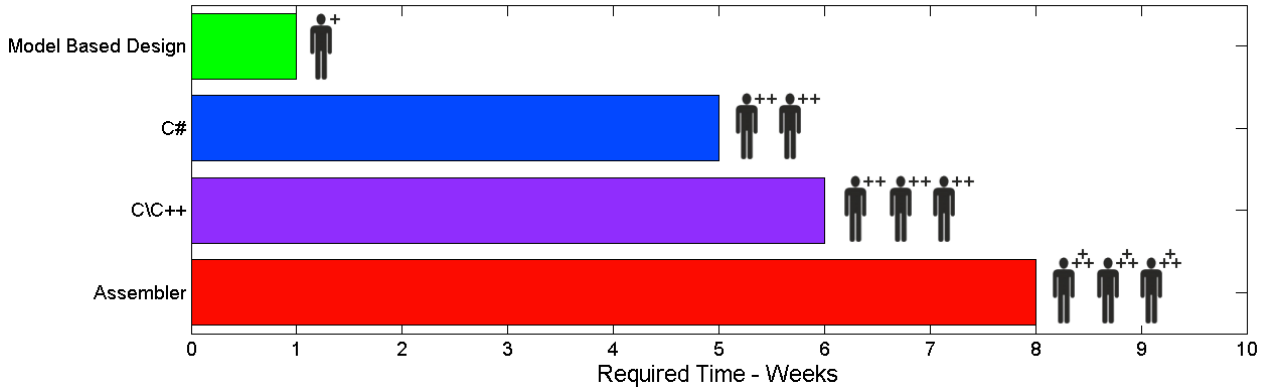


Ilustración 17: Comparación de tiempos según el lenguaje de programación. Creada por Antonio Flores.

Además de MATLAB, MathWorks desarrolla y comercializa Simulink, un producto concebido inicialmente para la simulación de sistemas dinámicos no lineales. Actualmente puede desempeñar las funciones de un lenguaje de programación completo basado en gráficos. Ofrece un conjunto de bloques predefinidos que se pueden combinar a fin de crear un diagrama de bloques detallado de un sistema. Pueden interpretarse como una pieza configurable modular de código (Ver *Ilustración 18*). Las herramientas de modelado jerárquico, gestión de datos y personalización de subsistemas permiten representar hasta los sistemas más complejos de forma concisa y precisa.

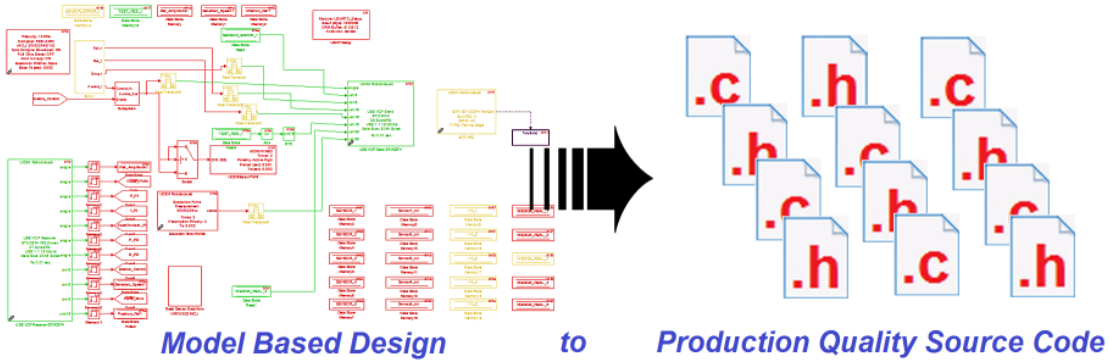


Ilustración 18: Código que genera Matlab al usar programación de bloques.

Imagen cortesía de Antonio Flores.

Simulink Library Browser contiene una biblioteca de bloques empleados habitualmente para modelar un sistema. Esto incluye: bloques dinámicos continuos y discretos; como *Integration* y *Unit Delay*, bloques de algoritmos; como *Sum*, *Product* y *Lookup Table* y bloques estructurales; como *MUX*, *Switch* y *Bus Selector*. Además, es posible crear funciones personalizadas mediante el uso de estos o a través de la incorporación de código manual de MATLAB en el modelo mediante el bloque *Function*. Los de creación propia se pueden almacenar en sus propias bibliotecas dentro de Simulink *Library Browser*.

Para que no haya problemas de incompatibilidad con nuestro sistema de ARCP, se requiere la instalación de software adicional a la biblioteca del navegador de Simulink. Consiste en dos conjuntos de bloques. El primer conjunto funciona como software de base cubriendo muchos periféricos de E/S; está proporcionado por una tercera empresa llamada Aimagin Ltd. (Aimagin, Bangkok, Tailandia).

El segundo de estos conjuntos de bloques, que funciona como una extensión de la anterior, ha sido desarrollado en su totalidad por la UC3M RoboticsLab y es el núcleo del sistema de RCP propuesto. Incluye enlaces de alta velocidad de intercambio de datos de bus serie universal (USB) con MATLAB/Simulink y características avanzadas, como el soporte multitarea en tiempo real, soporte completo de E/S periféricas y algunas optimizaciones de rendimiento muy especializadas en el código generado.

Hasta obtener un programa ejecutable para cargarlo al microcontrolador, son necesarias una serie de etapas (Ver *Ilustración 19*). Comenzamos desarrollando el algoritmo con los bloques de Simulink. Dicha herramienta genera automáticamente la traducción de los gráficos a un lenguaje textual. Después, necesita también incluir un compilador para traducir el código fuente de C a lenguaje ensamblador. Hay opción a seleccionar tres entornos diferentes del compilador, la libre, GNU-ARM, y las comerciales, Keil uVision (ARM Alemania, Grassbrunn, Alemania) y IAR Ewarm (IAR Systems AB, Uppsala, Suecia). En mi caso elegimos e instalamos Keil uVision 5.

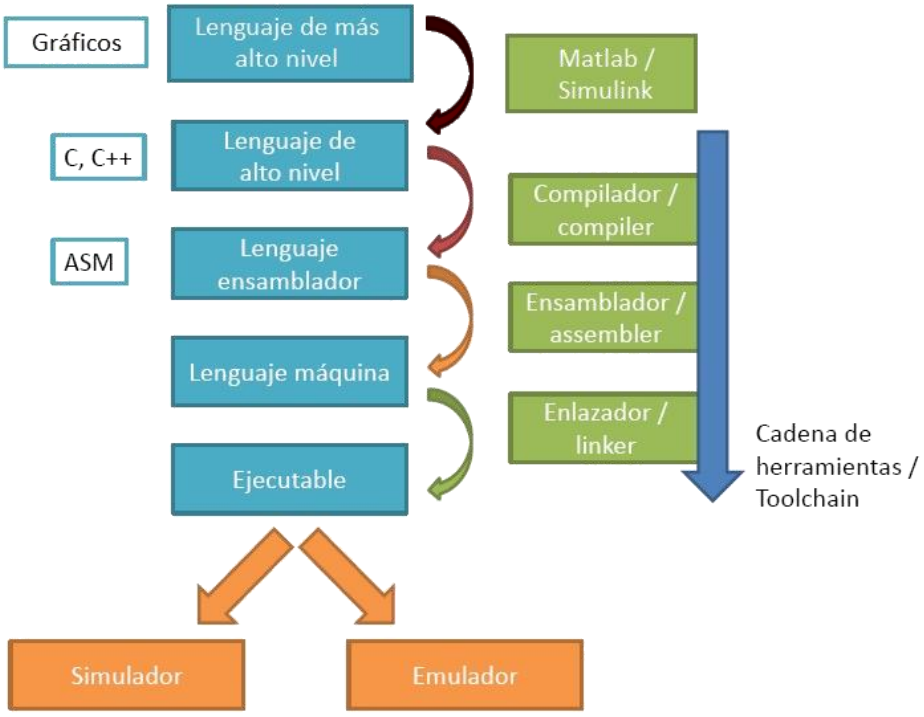


Ilustración 19: Niveles de abstracción en la programación. Fuente: [18].

Una vez que hemos superado los pasos previos, conectamos directamente el hardware RCP a un puerto USB del ordenador, y con un solo "click" del usuario, Simulink transfiere automáticamente el modelo.

-Programación

Tras analizar los objetivos y disponer los recursos necesarios, nos planteamos cuál será el método más idóneo para desarrollar el algoritmo. Comenzamos plasmando la idea general en un diagrama de flujo para sintetizar la solución. Posteriormente seleccionamos los bloques que cumplan las funciones correspondientes a cada instrucción. A su vez, comentaremos su fundamentación y propondremos alternativas.

Planteamos una estrategia multitarea para un sistema de CPU mononúcleo. Significa ejecutar diferentes procesos de forma concurrente, es decir, comparten el tiempo. Sin embargo, al tener un procesador de un solo núcleo, las tareas no se pueden leer paralelamente. Hasta que no finaliza una instrucción no se da paso a la siguiente; esto es un comportamiento secuencial. La manera intermedia de hacer que el sistema sea aparentemente concurrente es establecer prioridades y permutar el procesamiento de cada una de ellas en los diferentes instantes de tiempo.

Un elemento esencial del control es manejar la comunicación entre el microcontrolador y el robot. De forma general, al trabajar con un tipo de procesador diseñado para cumplir aplicaciones específicas, es determinante conocer las conexiones de dicho dispositivo con su fin externo. Podemos clasificar los sistemas informáticos teniendo en cuenta su interacción con el resto del entorno [21]. El evento no es subrutina, y puede ocurrir en cualquier instante de tiempo, pero la respuesta del sistema tiene que ser siempre la misma. Esto es responsabilidad del programador:

• Sistemas reactivos: son aquellos que siempre interactúan con el exterior, de tal forma que la velocidad de operación del sistema deberá ser la velocidad del entorno exterior. Su ejecución se debe a la detección de un evento de entrada. Puede ser a intervalos regulares de tiempo o independiente a través de interrupciones. La mayoría de sistemas empotrados suelen ser reactivos, ya que su diseño está adaptado a este tipo de problemas. De hecho, son más comúnmente conocidos como sistemas de tiempo real porque impone que el programa de control actúe cada determinado tiempo y no pueda rebasar los plazos temporales entre iteraciones.

• Sistemas interactivos: son aquellos que siempre interactúan con el exterior, de tal forma que la velocidad de operación del sistema deberá ser la velocidad del propio sistema empotrado. Se ejecutan continuamente, solo terminan cuando se les fuerza a ello. Mientras, su estado y salidas evolucionan si sus entradas cambian.

• Sistemas transformacionales: son aquellos que no interactúan con el exterior, únicamente toma un bloque de datos de entrada y lo transforma en un bloque de datos de salida, que no es necesario en el entorno. Se ejecutan una única vez.

En nuestro trabajo nos resultan relevantes las dos primeras. La fundamentación del sistema en tiempo real se ha desarrollado desde el primer momento ya que es un objetivo importante en el control. Por otro lado, el interés de la ejecución en tiempo continuo se basa en la combinación con procesos de tipo reactivo.

Si se quiere que una tarea interrumpa su ejecución para dar prioridad a un sistema de tiempo real necesitamos utilizar procedimientos interactivos. Además, para sistemas RCP es especialmente ventajoso si no conocemos el tiempo exacto entre procesos. Sobre todo si se van a recibir grandes cantidades de datos, evitamos la saturación del programa. Esto sería posible si programásemos la recepción con la primera técnica.

Teniendo en cuenta lo anterior, dividimos nuestro problema en tres cometidos. En primer lugar, definimos la recomposición del dato en tiempo continuo. Estará activa siempre que no haya ninguna otra tarea ejecutándose. Por defecto, el sistema ARCP establece una tarea de inactividad (*IDLE task*) para ahorrar energía cuando no es necesario atender ningún proceso [22].

Sin embargo, decidimos aprovechar este tiempo de forma que establecemos dicha función como una tarea de baja prioridad. En segundo lugar, tenemos que enviar el dato. Lo hacemos en tiempo real a la frecuencia de 1 kHz. Por último, la estrategia pensada para la captura de los mismos será la interrupción.

Se tiene que realizar al detectar los flancos de subida de la señal de reloj como se muestra en el manual de los encoders (Ver *Ilustración 20*). Incluye un pequeño tiempo de retraso t2 que no vamos a tener en cuenta para la programación. La precisión en la medición del tiempo abarca desde 100 ns hasta microsegundos, configurable por el usuario.

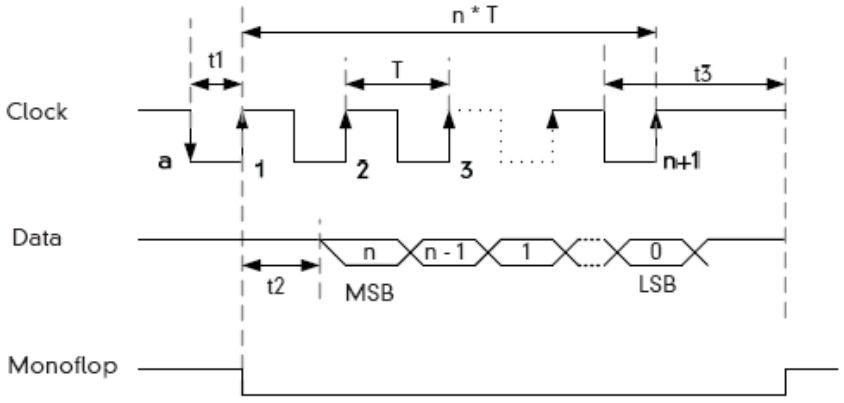


Ilustración 20: Ejecución de las tareas en el tiempo de mayor a menor prioridad.

Fuente: [23].

El criterio de prioridad es inverso a la explicación de las mismas. Siguiendo una estructura lógica, la captura de los datos se tiene que hacer con antelación a su recomposición. Por eso es la primera. También el envío es precedente al registro final de las muestras. Se puede entender más fácilmente observando la *Ilustración 21*.

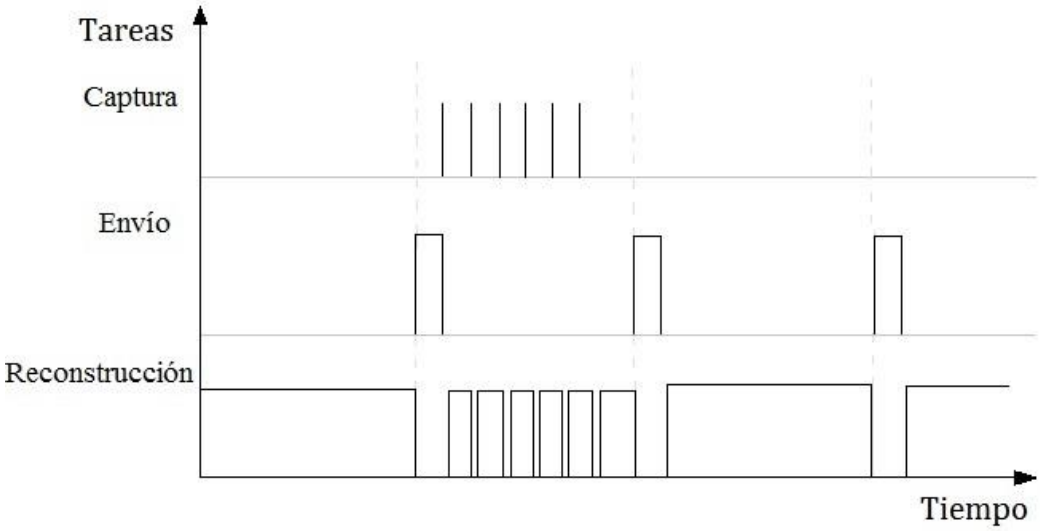


Ilustración 21: Ejecución de las tareas en el tiempo de mayor a menor prioridad.

Una interrupción es un evento hardware producido por una señal externa al micro o internamente por un periférico (temporizador, puerto, etc.). Este suceso queda registrado y pendiente de que se procese. Se trata en una rutina de atención a la interrupción (RAI). Cuando el sistema detecta una interrupción, la ejecución principal se suspende temporalmente para atender a las instrucciones incluidas en su rutina de servicio. Nos permite atender hechos en tiempo real, corregir errores, procesar datos de entrada…

Es necesario que tengamos en cuenta una serie de aspectos a la hora de programar interrupciones. El programa principal suele encargarse de algunas tareas, y se paraliza para atender la rutina de servicio. Por este motivo los bucles deben situarse en el programa principal y no dentro de esta. Debe tener un tiempo de ejecución muy corto.

Cuando el micro atiende una interrupción el contador de programa, que indica la dirección de memoria de la próxima instrucción a ejecutar, se guarda en la pila. El programador puede usar la pila para salvar datos temporales, pero su tamaño es limitado. Para no tener errores en la RAI no pueden aparecer subrutinas anidadas. Tampoco se le pueden pasar variables, es necesario que sean globales o compartidas con el *main*, peor hay que racionalizar el uso de la memoria.

Hay otra forma de comunicarse con periféricos y dispositivos externos: la espera activa o *polling*. Consiste en permanecer atentos al cumplimiento de una condición. El evento permitirá continuar la ejecución cada vez que ocurra, pero mientras el tiempo corre y la pregunta se comprueba constantemente. Además, no hay una rutina específica como sucede con el método de interrupciones (Ver *Ilustración 22*). Como en nuestra aplicación tenemos delimitado el rango de tiempo en que queremos hacer el proceso, la mejor manera de ejecutarlo es por interrupción. La espera activa es útil cuando no conocemos exactamente cuándo va a suceder el evento.

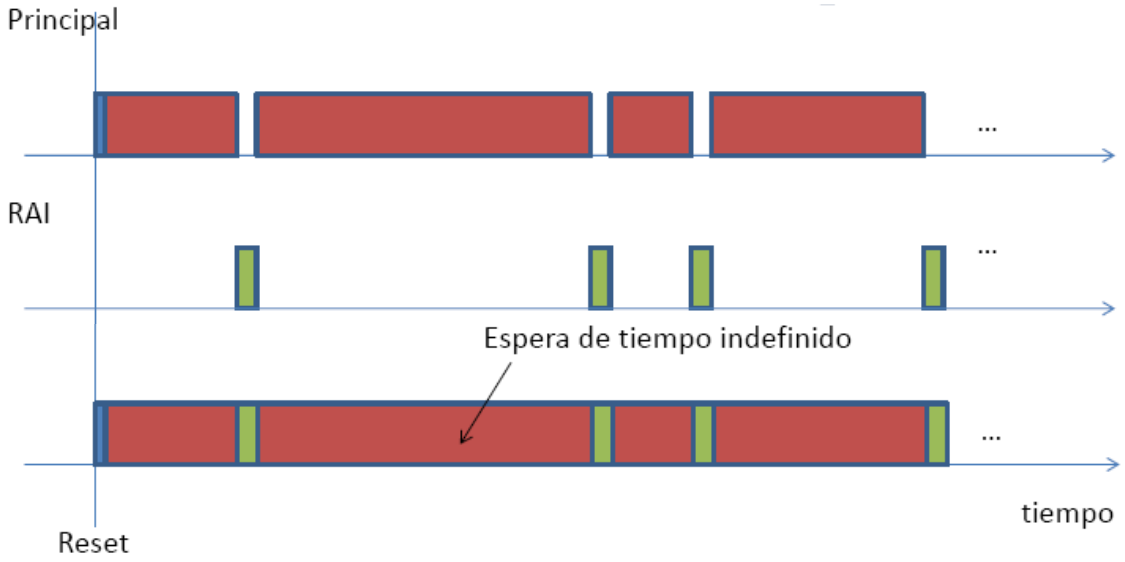


Ilustración 22: Diferencias entre interrupción (superior) y espera activa (inferior).

Fuente: [18].

Un aspecto importante en las prácticas de control es la capacidad de medir y manejar el tiempo. Analizamos los temporizadores que incluye el MCU así como los diferentes bloques desarrollados por la UC3M para el sistema ARCP (Ver *Ilustraciones 23* y *24*).

Hay diez temporizadores de uso general sincronizables integrados en el STM32F4. Existen diferentes funcionalidades [24]. Los básicos (TIM6 y TIM7), se utilizan principalmente como *trigger* para conversiones DAC y la generación de formas de onda. También sirven como base de tiempo genérico con resolución de 16-bit y para solicitar una generación independiente de acceso directo a memoria (DMA en inglés).

Los temporizadores de control avanzado pueden ser vistos como generadores de PWM de tres fases multiplexados en 6 canales. Tienen una compatibilidad tan completa como los de uso general. Sus 4 canales independientes se pueden utilizar para: captura de entrada, comparadores de salidas, generación de PWM y como salida en forma de impulso (*one-pulse mode output*).

Si se configura como estándar de 16 bits, funciona igual que los de propósito general. Si se configura como generador de PWM de 16 bits, tienen la capacidad de modulación completa (0-100%). Puede trabajar junto con los temporizadores TIMx a través de la función de enlace para la sincronización o encadenamiento de eventos.

El temporizador *Sys Tick* está dedicado a los sistemas operativos en tiempo real, pero también podría ser utilizado como un contador descendente estándar. Tiene un contador descendente de 24 bits, capacidad de recarga automática, sistema para enmascarar la generación de interrupciones cuando llegue a 0 y fuente de reloj programable.

Contiene dos temporizadores llamados *watchdog* que son un organismo de control para restablecer el dispositivo cuando se produce un problema. Hay dos tipos: independiente o de ventana. Ambos se basan en un contador descendente. Se pueden utilizar como funcionamiento libre (*free-running timer*) para la gestión de tiempo de espera. Se distinguen en la resolución y aplicaciones.

El primero de ellos es de 12 bits e incluye un pre-escalador de 8 bits. Funciona independientemente del reloj principal, puede operar en los modos de parada y reposo (*standby*). Es configurable por hardware o software a través de los bytes de opciones. El segundo es de 7 bits. Es registrado desde el reloj principal. Tiene una capacidad de interrupción de alerta temprana y el contador se puede congelar en modo de depuración.

Timer type	Timer	Counter resolution	Counter type	Prescaler factor	DMA request generation	Capture/ compare channels	Complementary output	Max interface clock (MHz)	Max timer clock (MHz)
General purpose	TIM2, TIM5	32-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	No	42	84
	TIM9	16-bit	Up	Any integer between 1 and 65536	No	2	No	84	168
	TIM10, TIM11	16-bit	Up	Any integer between 1 and 65536	No	1	No	84	168
	TIM12	16-bit	Up	Any integer between 1 and 65536	No	2	No	42	84
	TIM13, TIM14	16-bit	Up	Any integer between 1 and 65536	No	1	No	42	84
Basic	TIM6, TIM7	16-bit	Up	Any integer between 1 and 65536	Yes	0	No	42	84
Advanced-control	TIM1, TIM8	16-bit	Up, Down, Up/down	Any integer between 1 and 65536	Yes	4	Yes	84	168

Ilustración 23: Características de los temporizadores existentes en el STM32F4.

Fuente: [24].

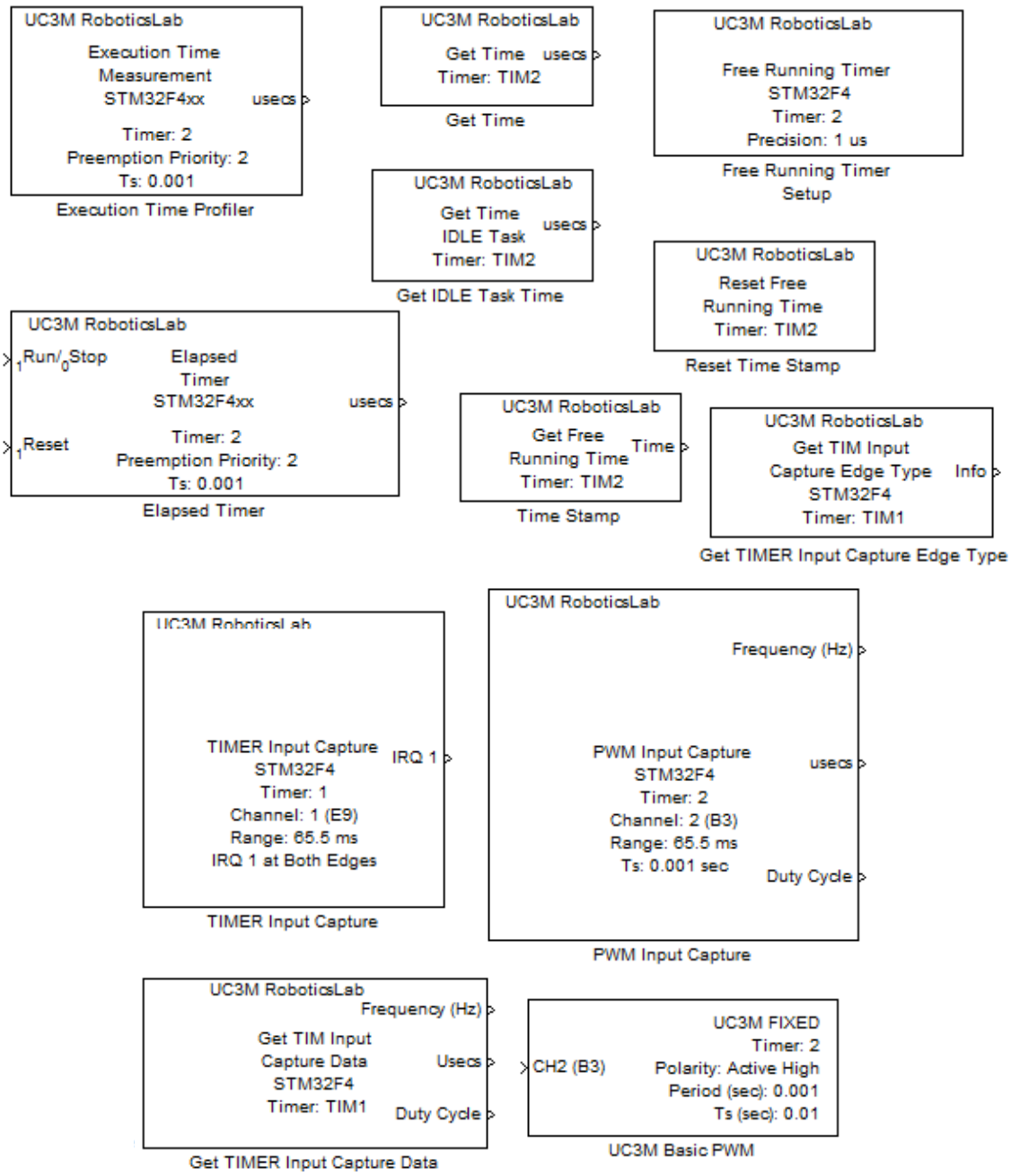


Ilustración 24: Bloques creados para SIMULINK® del sistema ARPC para gestionar los *timers*.

Fuente: [12].

▫ <u>Algoritmo</u>

Aplicando los conocimientos expuestos anteriormente, creamos un algoritmo en Simulink que nos permita controlar un encoder. En la rutina principal (*main*) del programa haremos la comprobación de si tenemos 17 capturas, si no se cumple seguimos capturando. Cuando sea cierto, entonces recuperamos el dato. Se tiene que registrar en el orden correcto. El subprograma de captura se desarrolla dentro de cada interrupción. Tomamos muestras en los flancos de subida del reloj.

También, necesitaremos un contador para evitar problemas de seguridad o diferencias de tiempos en la alimentación de las unidades de control frente a la de potencia. Si durante un tiempo de aproximadamente 100µs no hay ninguna interrupción externa, entonces reseteamos el número de capturas a cero. Se pone a cero cada vez que haya una interrupción.

Empezamos por analizar una de las articulaciones de la muñeca del robot, ya que como hemos mencionado anteriormente en el hardware hay diferencias importantes que afectan a la programación. Esto implica que trabajamos con 18 bits. La explicación del algoritmo se va a clasificar con un método deductivo. Comenzamos por mostrar la visión general del programa completo (Ver *Ilustración 25*). Posteriormente, lo dividiremos en etapas para una comprensión detallada de cada bloque de instrucciones.

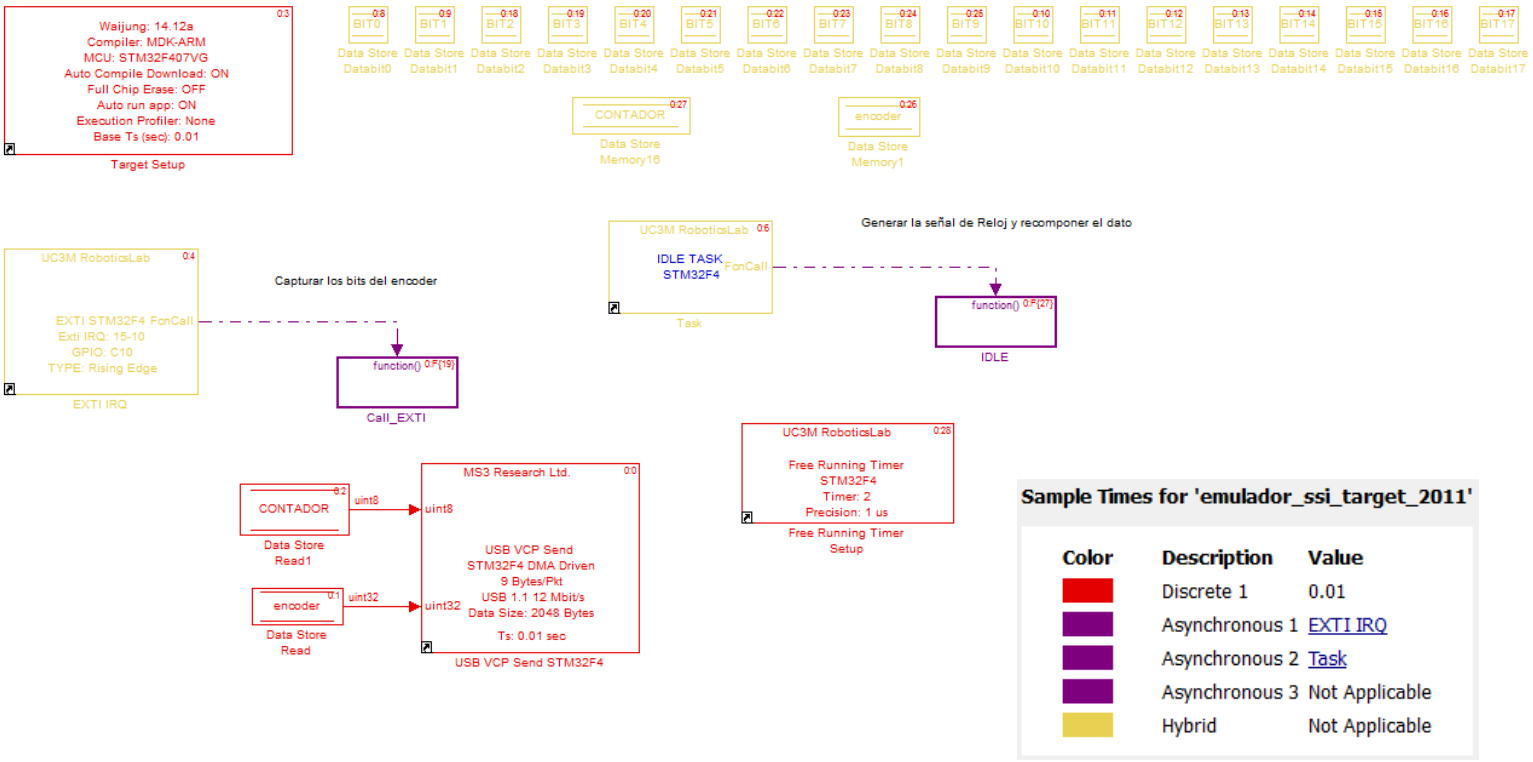


Ilustración 25: Programa de Simulink para una articulación de la muñeca.

Podemos observar en la imagen superior la diferente tonalidad de cada bloque. Esto se debe a una propiedad de Simulink para distinguir en qué tipo de procesos estamos trabajando. Extrayendo la información proporcionada por el cuadro de leyenda de colores vemos que conviven comunicaciones asíncronas, síncronas y sistemas discretos.

Los procesos síncronos son aquellos que se llevan a cabo en intervalos conocidos y constantes, como por ejemplo el tic-tac de un reloj [25]. Se dice que están sincronizados cuando ambos coinciden. La sincronía es de vital importancia en el tratamiento de aparatos electrónicos. Cada parte de un procesador, por ejemplo, se comunica con las otras mediante datos que incluyen enormes cadenas de bits.

Estos elementos deben estar sincronizados, para lo cual se han diseñado protocolos que les permiten identificarse mutuamente y transmitirse la información, algunos más elaborados que otros. Esta técnica se usa para el envío remoto de información, mediante ondas, pero también entre periféricos de entrada y el procesador, o incluso en procesos internos de programa.

Los procesos de un programa que se ejecutan de forma independiente se llaman asíncronos, mientras que si son ejecutados en respuesta a otro proceso serán síncronos. De cierto modo podemos decir que la comunicación síncrona es simultánea y la asíncrona en diferido. Evidentemente la forma asíncrona es más fluida y por tanto permite mayores velocidades, siendo la más extendida en la actualidad, pues no requiere confirmación de recepción del emisor y elimina esperas innecesarias. Esta característica cumple con el objetivo de nuestro sistema ARCP de acortar tiempos.

Comenzamos seleccionando un bloque de arranque que define las características hardware del microcontrolador. Se encuentra en el apartado "STM32F4 *target"* de la librería "STM32F4 & NRF51 Basic Blockset" (Ver *Ilustración 26*). Se procesa en tiempo real. Es necesario para que la herramienta pueda transferir correctamente las tareas en la placa.

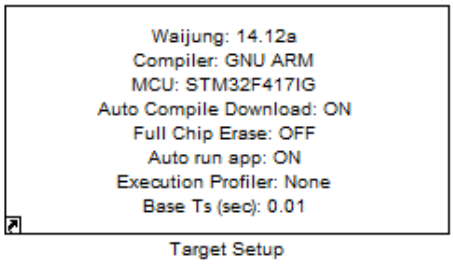


Ilustración 26: Bloque de configuración del microcontrolador.

Luego, declaramos las variables necesarias (Ver *Ilustración 27*). Situamos los bloques en la parte superior del programa. Tanto para facilitar una rápida interpretación gráfica de los recursos usados, como para evitar errores del programa las ponemos de forma independiente. Al contrario que en siguientes tareas donde será preferible clasifiicar los bloques en un subsistema que los englobe según la aplicación. Este aspecto facilita el entendimiento del algoritmo por una persona externa al programador ya que la información está más estructurada.

Asignamos un registro a cada bit correspondiente al dato que vamos a capturar. Una vez que se haya completado y recolocado se guardará en la varible llamada *encoder*. También necesitamos un contador para controlar el desarrollo de los procesos.

Es importante recordar que según cambiemos el enfoque de la programación se necesitarán un mayor número de variables. Si hiciesemos el control del brazo completo deberíamos registrar siete posiciones diferentes. Una correspondiente a cada sensor (encoder).

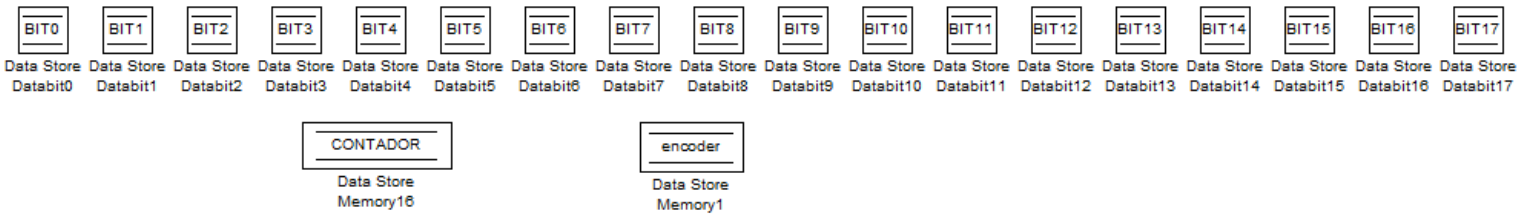


Ilustración 27: Variables del programa.

A continuación, se detalla la composición de cada tarea que forma nuestro planteamiento *multitask* al comienzo de este capítulo de programación. Se seguirá respetando la prioridad de las mismas para la explicación.

Inicialemnte seleccionamos el bloque para interrupciones externas de los periféricos disponibles en la *toolbox* "STMF4 v3.0 Beyond Control" creada por el UC3M RoboticsLab e incluida en las bibliotecas de Simulink (Ver *Ilustración 29*). Configuramos el bloque principal para que se active con los flancos de subida de la señal de reloj. En el primer subsistema (I) se procede a capturar los datos. Utilizamos un switch case para escribir el bit correspodiente a cada dígito (*en la figura solo se muestra el primer bit de ejemplo) en su lugar correspondiente según indique el valor de la variable CONTADOR. En el segundo (II), incrementamos el valor del contador y también reseteamos el timer definido en *real time*. Este proceso es asíncrono.

El temporizador se arranca fuera de este gran subsistema de interrupción. Se encuentra dentro de la misma librería que el EXTI, pero en el epígrafe de *timers*. Es de funcionamiento libre y se ejecuta en tiempo real (Ver *Ilustración 28*).

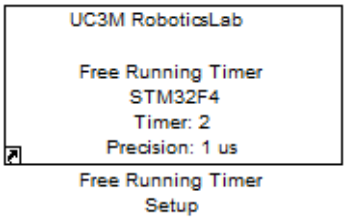


Ilustración 28: Temporizador de funcionamiento libre en tiempo real.

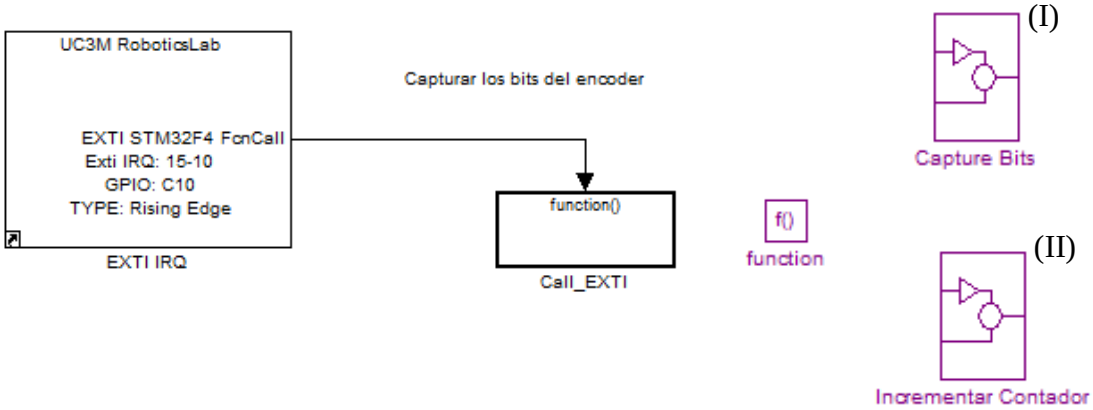


Ilustración 29 - A: Bloque principal y subsistemas que se ejecutan en la interrupción.

(I)

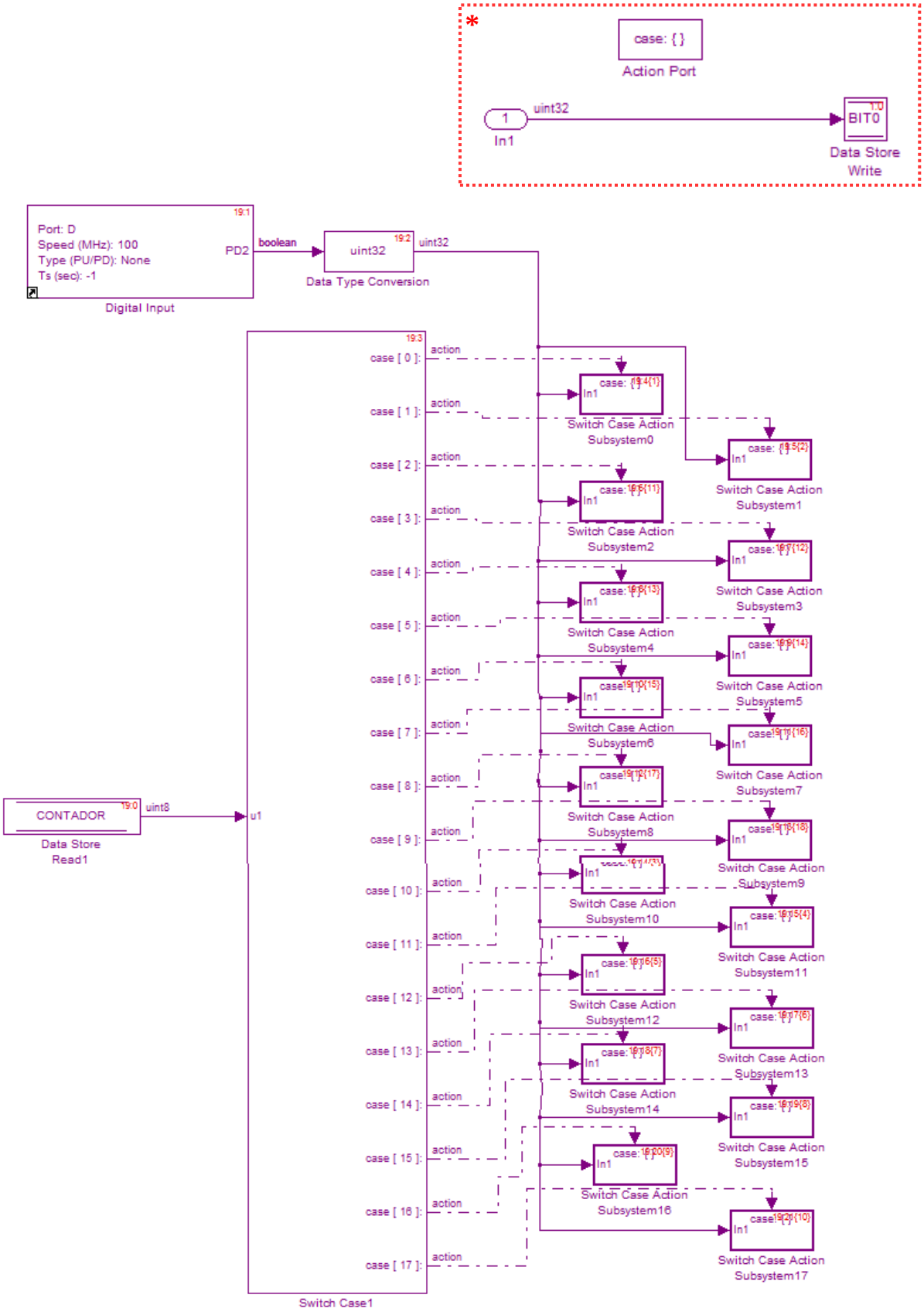


Ilustración 29 - B: Detalle del primer subsistema de captura de bits.

(II)

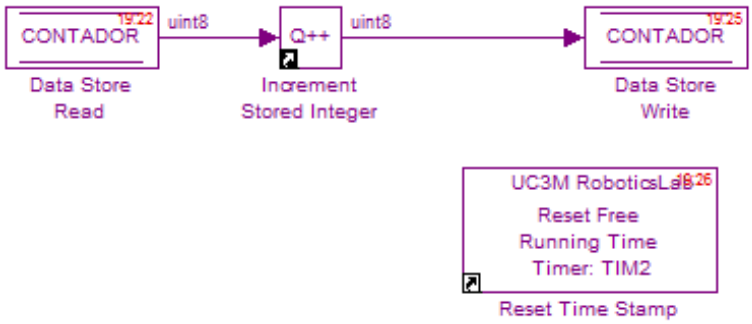


Ilustración 29 - C: Detalle del segundo subsistema para el incremento del contador.

En tiempo real enviamos el dato a frecuencia de 1kHz. Ajustamos el parámetro de tiempo de muestreo (Ts) para cumplir esta condición (Ver *Ilustración 30*). Transmite grandes paquetes de datos (hasta 16 KB a la vez) utilizando el protocolo USB como un puerto de comunicación virtual (VCP en inglés, *Virtual COM Port*). Sólo es compatible con datos en formato binario.

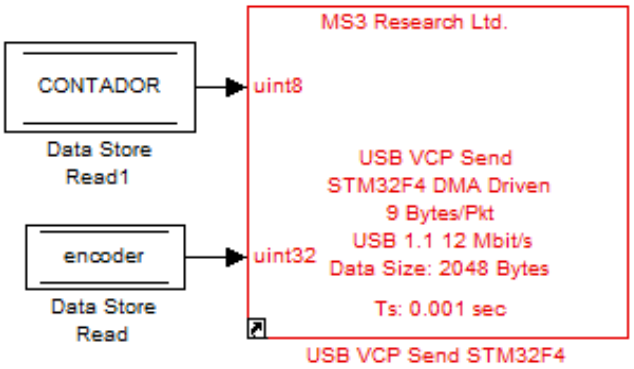


Ilustración 30: Bloque para el envío de datos al microcontrolador.

Para concluir el programa multitarea determinamos el subsistema del proceso en tiempo continuo (Ver *Ilustración 32*). En el esquema propuesto al comienzo, fundamentamos porque se tiene que recolocar el dato. Necsitamos que la estructura del valor que indica la posición final de la articulación sea lógica para poder extraer información de ella.

Incluimos un bloque *if* para poder comprobar la condición de que ya tenemos todos los bits capturados. Esto se traduce en que el CONTADOR valga más de 17 ya que se incrementa con cada bit y son un total de 18. Cuando sea cierto, entonces en su bloque de acción desplazamos los bits para que estén en el orden correcto. Además, cuando llega a esta situación tiene que volver a empezar la cuenta, es decir lo reseteamos. Por último, registramos el dato a través de una puerta OR en una variable llamada encoder.

Por otro lado, con la mayor prioridad dentro de la tarea, reseteamos el contador como medida de seguridad mediante la comprobación de que el valor del timer supere los 30 µs. Con el objetivo de tener un sistema robusto fijamos dicho tiempo para que esté comprendido entre el espacio entre pulsos, y la duración de un tren completo de 18 (Ver

Ilustración 31). Es importante este bloque para controlar y evitar fallos en la lectura de datos por falta de alimentación debido a apagados y encendidos repentinos o cualquier error del sistema.

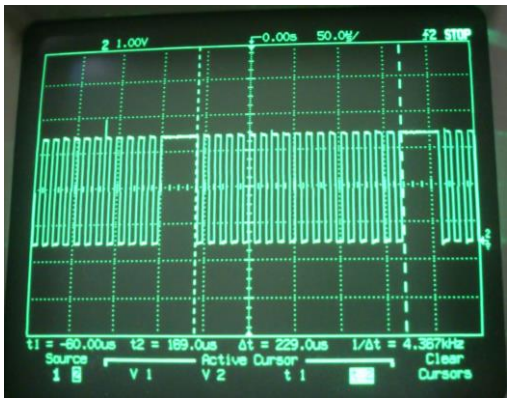


Ilustración 31: Imagen de la señal de reloj en el osciloscopio.

Después, generamos una señal de reloj para emular la salida que obtendríamos del driver si su funcionamiento fuese el correcto. Sin este conjunto de bloques definidos no podíamos avanzar el proyecto porque a través de la detección de sus flancos se procede a la captura. La prioridad es intermedia, ya que se sitúa después del reset de seguridad, pero antes de la recomposición del dato. Su diseño consiste en forzar el nievel lógico de uno al principio y al final, utilizando su salida digital correpondiente en la placa. Repetimos el subsistema que se encarga de estas operaciones.

Para emular la forma de una señal de reloj tenemos un bloque para generar 18 pulsos. Lo hacemos con un bucle while donde indicamos que el número de iteraciones sea 18. Controlamos el tiempo que está a uno o a cero intercalando unas esperas ajustadas para obtener la frecuencia deseada de 1kHz. Por último, para diferenciar una secuencia de la siguiente se requiere añadir un tiempo de espera (*t3*) como nos indica el manual del encoder y podemos observar en la *Ilustración 20*. Luego volvemos a repetir el ciclo.

La colocación de los procesos se hace de forma vertical, comenzando por la parte superior hasta el final de la hoja de programa, de mayor a menor prioridad. En las imágenes se muestran con la misma situación real en que se posicionan en Simulink.

El bloque if * antes de la última espera lo añadimos posteriormente para forzar la lectura de un dato sin conectar el encoder. En este ejemplo ponemos el número 5 combinando los tres bits finales con su codificación en binario: 101. Para dar el valor a la salida dato utilizamos la misma estructura de constante y señal digital de salida anteriores, pero cambiamos el pin asociado.

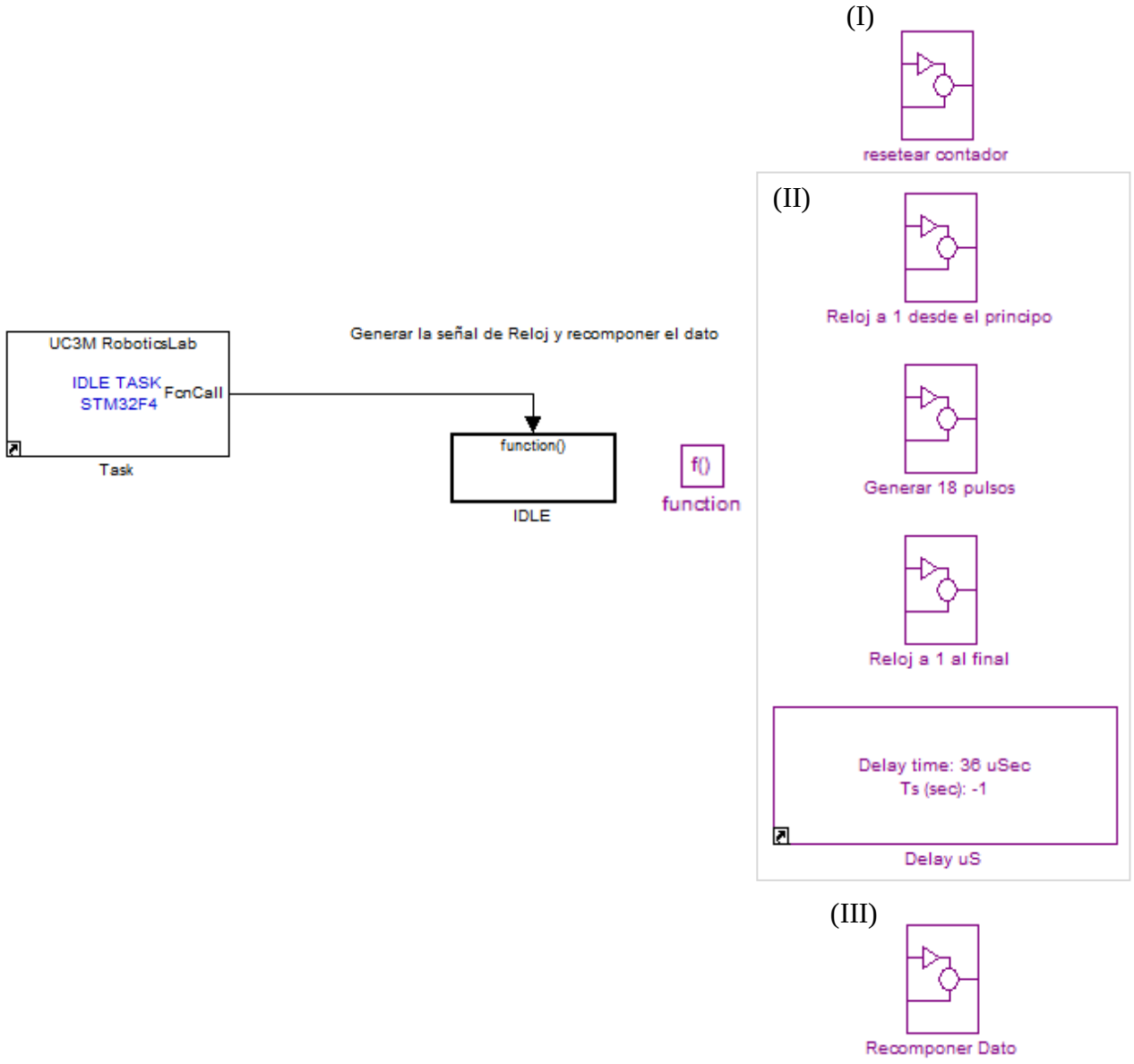


Ilustración 32 - A: Bloque principal y subsistemas de la tarea en tiempo continuo.

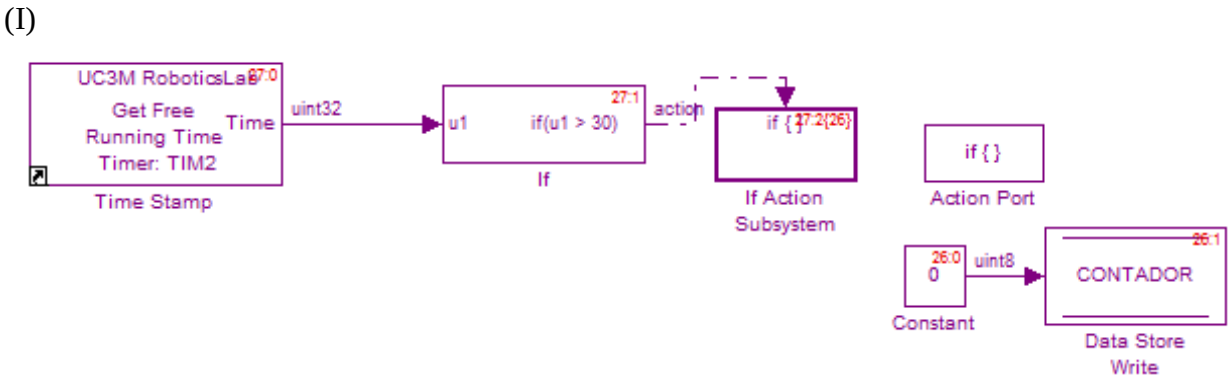


Ilustración 32 - B: Subsistemas de reseteo del contador.

(II)

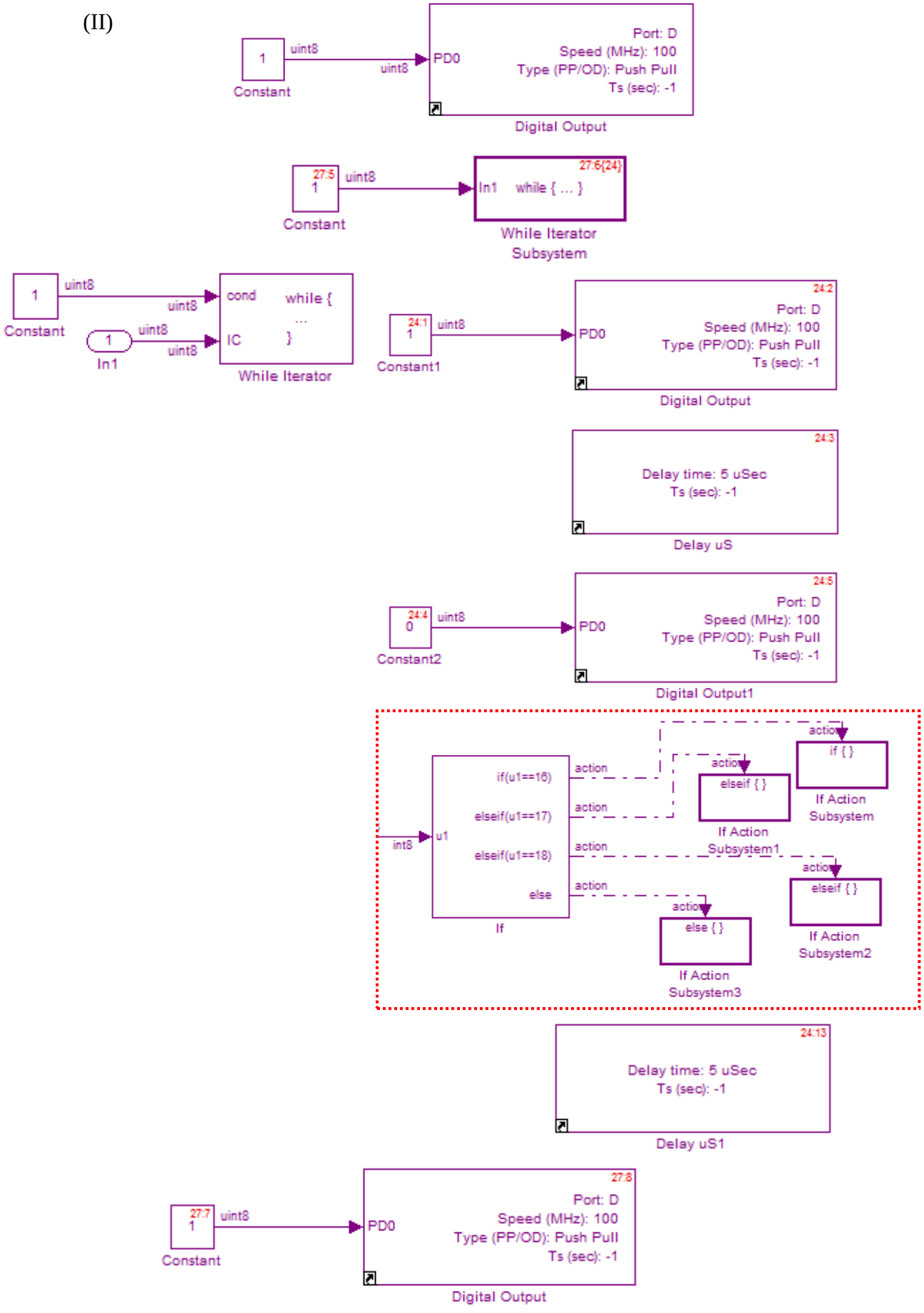


***Ilustración 32* - C:** Bloques para generar la señal de reloj.

(III)

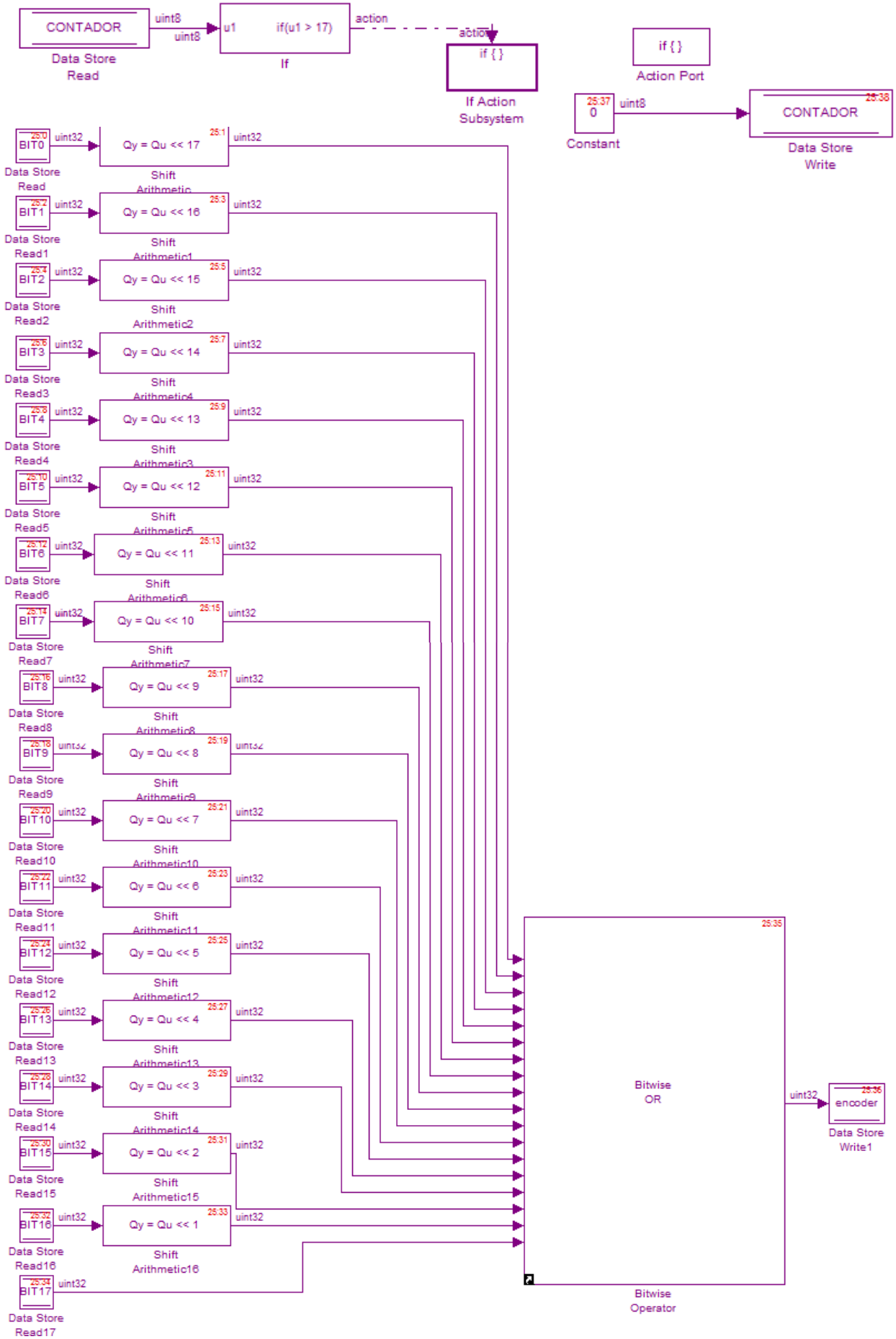


Ilustración 32 **- D:** Subsistema para recomponer el dato.

El algoritmo anterior le tomamos como esquema modelo para modificar nuestra estrategia. Tras realizar las pruebas oportunas y concluir que no podremos utilizar el driver, decidimos cambiar el enfoque de control. Se decide emular en un microcontrolador del mismo tipo que nuestro sistema ARCP el comportamiento de cuatro encoders con el procedimiento que se introdujo en la *ilustración 6*. Este programa es un dato de entrada para el TFG, ya que ha sido creado por el tutor Antonio Flores. La tasa de ejecución es de 1000 veces/s y la frecuencia de cada señal de 500kHz.

Ahora el problema consistirá en recuperar los datos de salida del dispositivo, distinguiendo cada sensor. Otro de los cambios propuestos en este método de control es que las articulaciones ahora corresponden a la parte superior del brazo. Son parte del hombro y el codo. Por tanto la resolución ahora es de 19 bits; propia del encoder DS-70.

A continuación mostramos el programa modificado (Ver *Ilustración 33*). Señalamos las diferencias respecto del algoritmo base. Al estudiar una aplicación con velocidades relativamente elevadas, en el que hay que atender las IRQ en muy poco tiempo, se ha de evitar la ejecución condicional (IF, SWITCH) pues éstas no logran procesarse en tiempos tan reducidos. Para evitar que surjan estos errores, buscamos otro tratamiento más rápido.

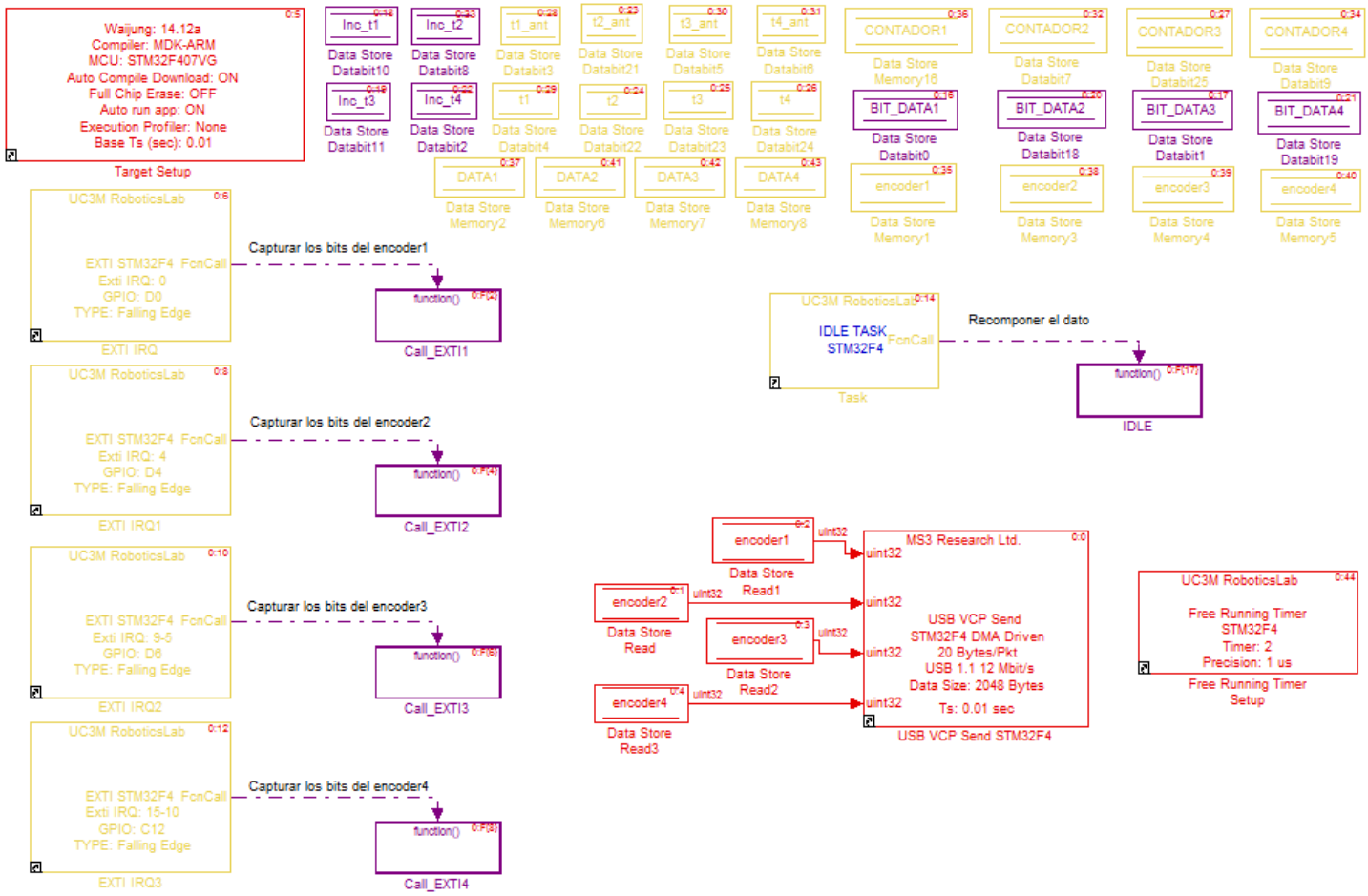


Ilustración 33: Programa definitivo en Simulink.

El modo de empleo del temporizador cambia, puesto que es el mismo dispositivo para el control de lecturas de las cuatro señales. Capturamos la medida temporal en variables diferentes para controlar la toma de datos de forma incremental. Necesitamos el valor anterior (t_x_ant) y el actual (t_x) para después trabajar con la resta (Inc_t_x). Compararemos dichas cifras para saber si necesitamos reiniciar el contador de bits debido a que no estamos registrando la información correctamente (Ver *Ilustración 34*).

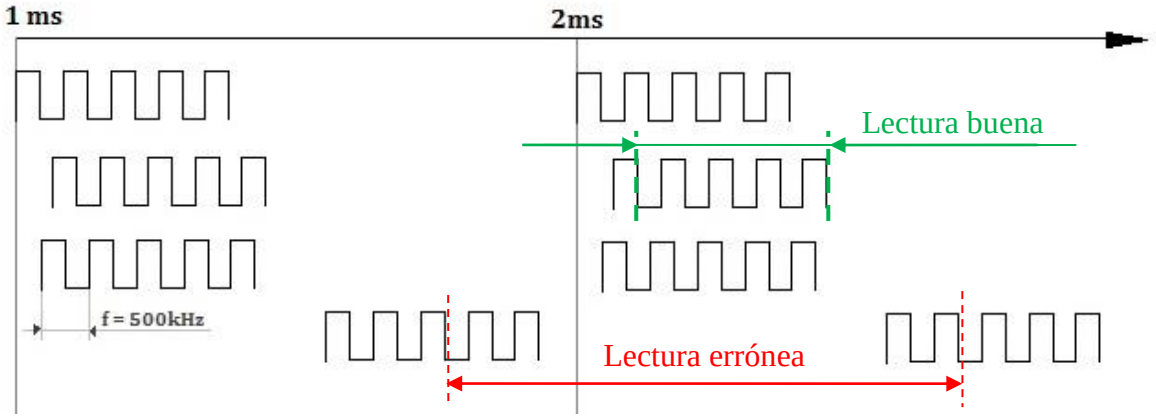


Ilustración 34: Control integrado de cuatro encoders.

Además de los añadidos anteriores, necesitamos declarar registros intermedios para guardar los datos ($DATA_x$). Se emplea para los cálculos obtenidos en las operaciones de la función incorporada en el subsistema de captación de bits dentro de la interrupción. Posteriormente copiaremos su contenido en $encoder_x$ que es el definitivo. (Ver *Ilustración 35*).

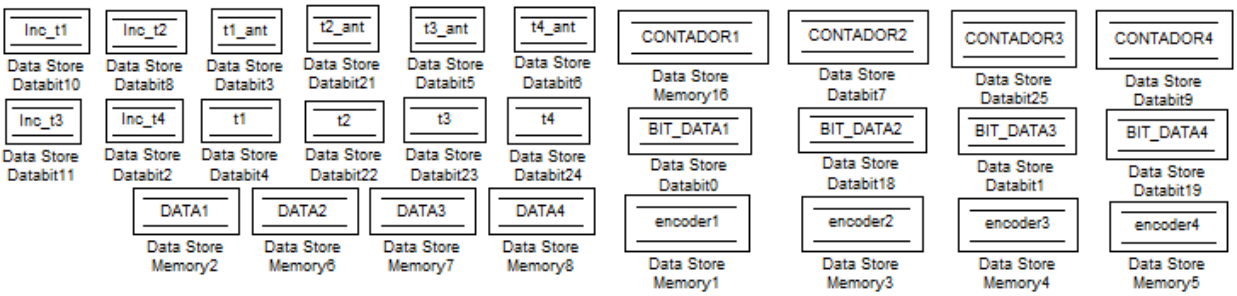


Ilustración 35: Variables utilizadas en el algoritmo.

Los flancos de reloj que provocarán la ejecución de las interrupciones externas son los de bajada. Como ya habíamos explicado, según los manuales deberían seleccionarse los de subida. Sin embargo, para ser coherentes con el algoritmo que emula los sensores tenemos que modificar este parámetro porque el tutor lo eligió al contrario. Para eliminar la ejecución condicional en la EXTI aprovecharemos la capacidad de Simulink para integrar funciones de Matlab.

Asimismo, nos aseguramos de que no se comparta ninguna fuente de interrupción; el bloque muestra que fuente de IRQ están usando. En la rutina de atención a la EXTI

realizamos tres procesos integrados en distintos subsistemas: captura de bits, incremento del contador y captura del tiempo. Se reproducen cuatro veces ya que el procedimiento es el mismo para todas las señales, pero necesitamos cambiar el identificador de las variables para saber que sensor estamos ejecutando (Ver *Ilustración 36*).

En primer lugar, capturamos los bits. Para ello copiamos la lectura proporcionada por la señal del encoder transformándola en una variable sin signo de 32 bits. Después, hay que utilizar un bloque de código .m para colocarlos en el orden correcto. Empleamos la función bitshift. Tiene como entradas el valor de cuenta y el bit que se quiere registrar para dar como salida el resultado oportuno ($data_x$).

El segundo subsistema es idéntico al algoritmo inicial simplificado. Se trata de incrementar el contador para poder controlar posteriormente si ya hemos completo el número de pulsos que componen un dato completo. Por último, como ya hemos comentado, guardamos la salida del *timer* en las variables correspondientes con el fin de evitar lecturas erróneas. El reseteo del contador se realiza fuera de la interrupción para evitar que el programa principal este desatendido mucho rato.

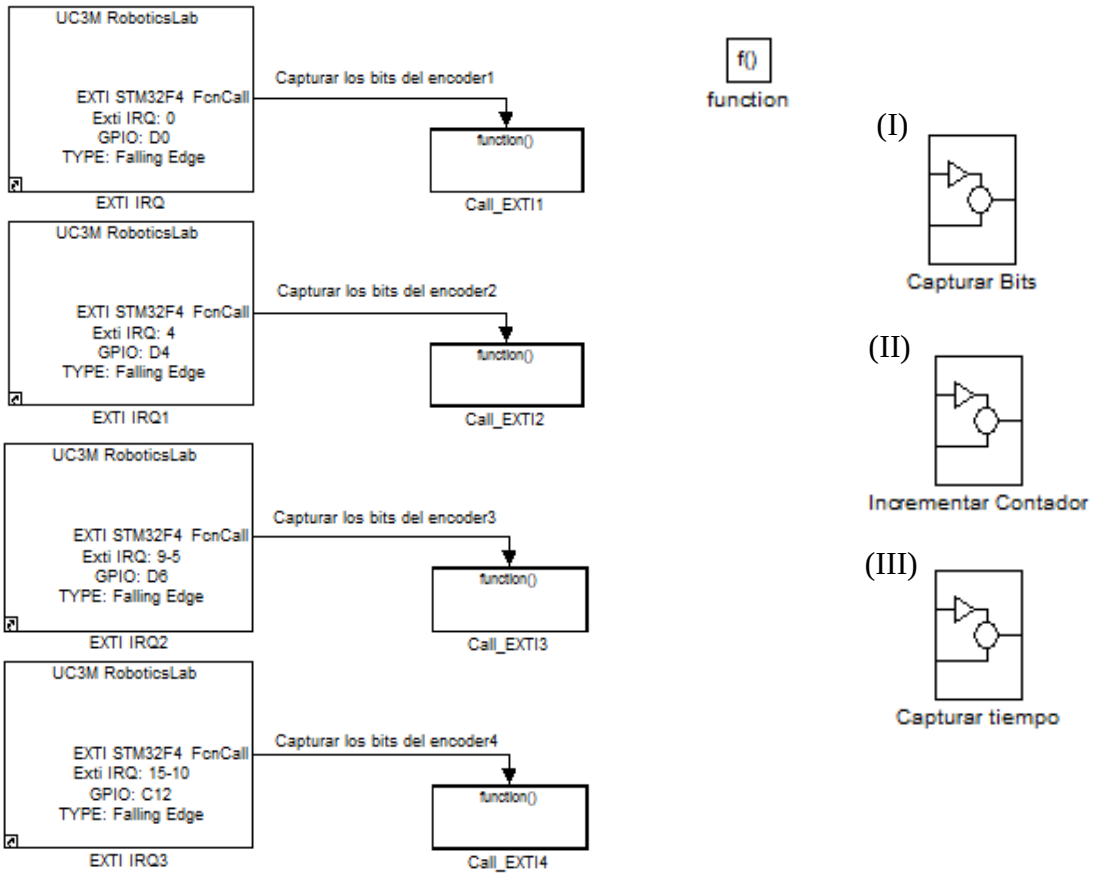


Ilustración 36 - A: Bloque principal y subsistemas que se ejecutan en la interrupción.

(I)

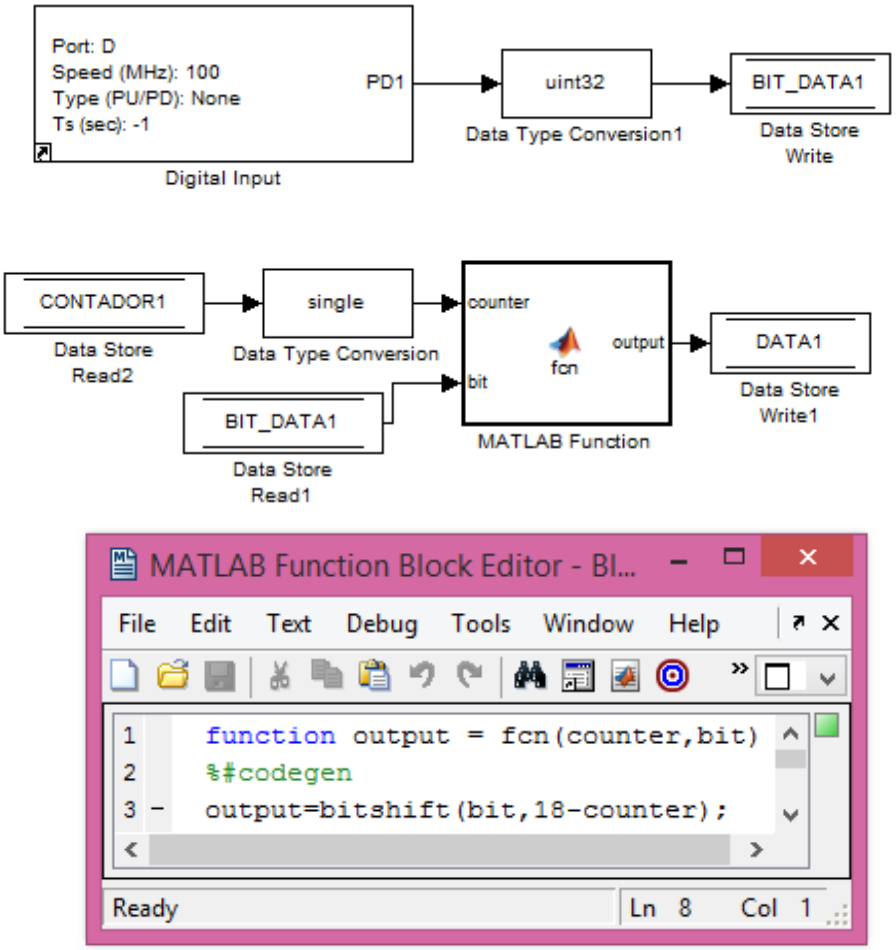


Ilustración 36 - B: Subsistema para la captar bits.

(II)

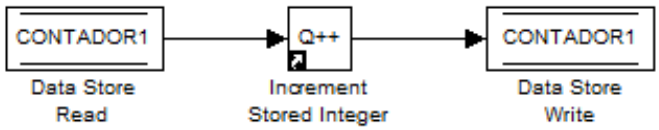


Ilustración 36 - C: Subsistema para incrementar el contador.

(III)

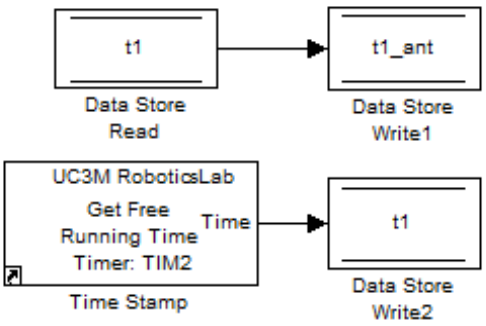


Ilustración 36 - D: Subsistema para captar el tiempo.

En la tarea de tiempo libre modificamos el reseteo de los contadores para que lea como condición la variable incremento de tiempo, en vez directamente la salida del temporizador. Como en la condición del *if* no se puede incluir una operación matemática, realizamos la resta y registro del incremento previamente.

Por otro lado, consultamos si se han obtenido los bits completos para poder dar por finalizada la recuperación de los datos. Esto se traduce a que el contador sea mayor que 18 porque el sensor produce una señal de tamaño 19, pero tenemos en cuenta que comienza en cero. Cuando se cumple, ponemos de nuevo a cero el contador y luego traspasamos los datos, ya ordenados en la interrupción externa, a una variable definitiva $encoder_x$ porque esta es la que enviaremos al microcontrolador por USB. Para acabar se inicializa la cuenta del temporizador (Ver *Ilustración 37*).

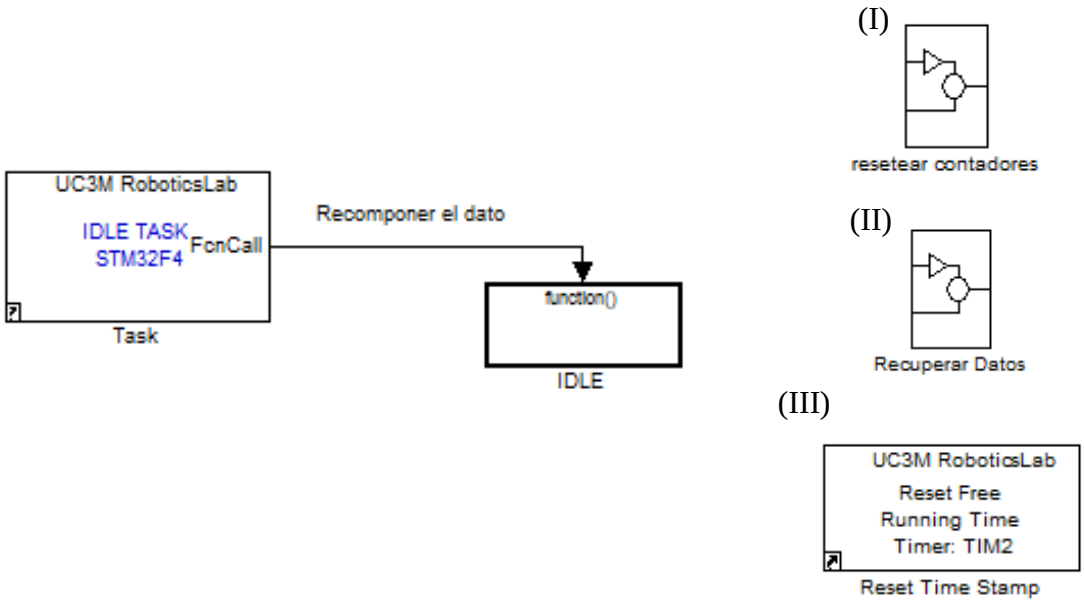


Ilustración 37 - A: Bloque principal y subsistemas del proceso de tiempo continuo.

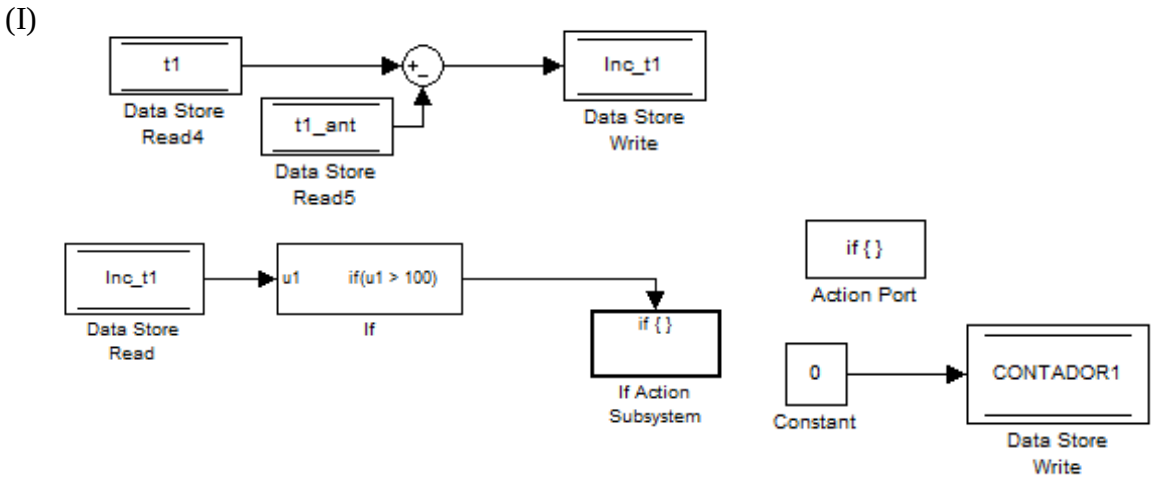


Ilustración 37 - B: Subsistema para resetear los contadores.

(II)

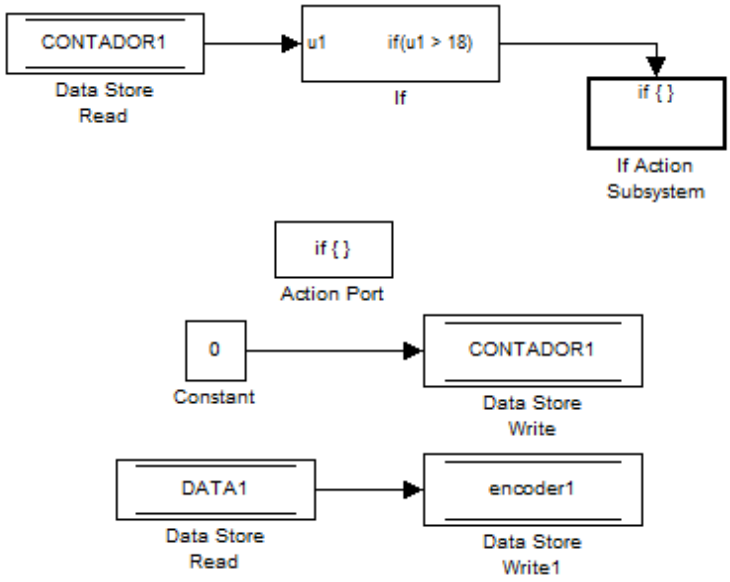
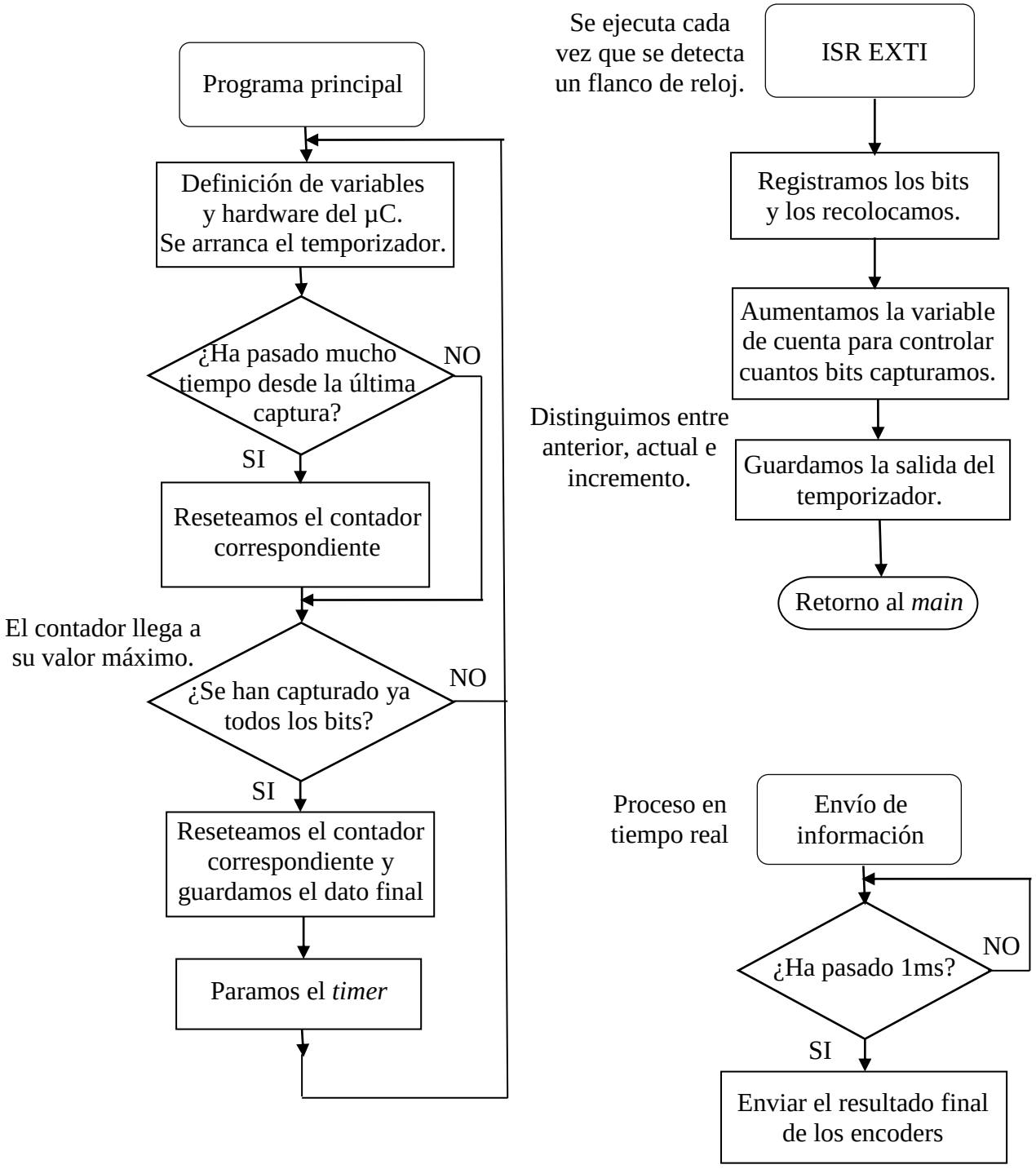


Ilustración 37 - C: Subsistema para recuperar datos.

Respecto de los bloques del temporizador y envío no es necesario versionar nada. Mandamos los cuatro registros de las variables encoder en tiempo real. Nos ahorramos los contadores que eran solo para facilitar las pruebas de forma visual. Si ha pasado 1 ms, se ejecuta automáticamente dicha tarea gobernada por el SysTick. Tiene más prioridad que la *idle task*.

-Diagrama de flujo

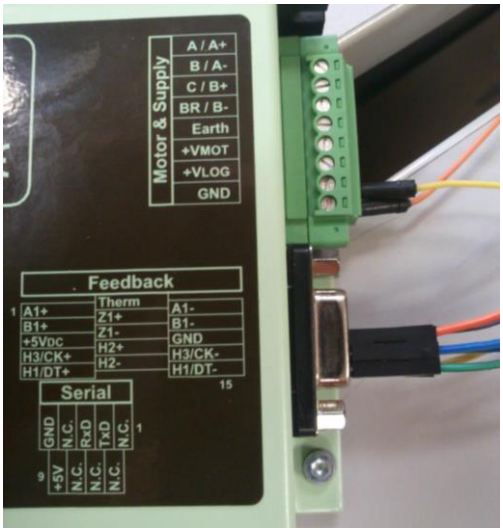
Se resumen de forma esquemática las instrucciones básicas del algoritmo definitivo desarrollado. De esta forma se facilita el entendimiento del mismo.



-Pruebas de funcionamiento

Es importante tener en cuenta la veracidad del trabajo realizado en todo momento. Por este motivo es imprescindible realizar pruebas que demuestren si las teorías y planteamiento del problema responden de la forma esperada. Según los resultados que vamos obteniendo sabremos si nos acercamos o alejamos del objetivo general aplicado al brazo robótico. Una vez detectados los errores haremos las modificaciones necesarias para subsanarlos.

Primero conectamos el driver a la alimentación (Ver *Ilustración 38*) y comprobamos que no haya ningún cortocircuito. *Pines Motor & Supply*: +VLog y GND. Podemos observar que la intensidad en reposo es elevada (300mA) lo que significa que consume mucho. Para enlazar el encoder utilizamos los pines de *feedback*. Nos encontramos con que la longitud de los cables del sensor es demasiado reducida, lo que nos obliga a empalmar otros cables para que lleguen correctamente a su destino.



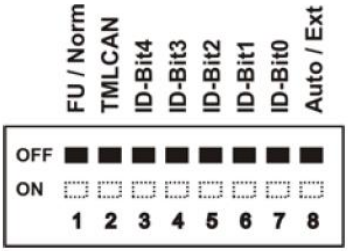
Señales	Color cable del encoder (Fuente: [14])	Color cable empalme	Número de pin en el driver (Fuente: [15])
Clk+	Gris	Azul	4
Clk-	Azul	Azul	14
Data-	Amarillo	Amarillo	15
Data+	Verde	Verde	5
GND	Negro	Morado	13
+5V	Rojo	Naranja	3

Ilustración 38: Conexiones del driver para comprobar su funcionamiento.

Utilizamos el programa Easy Motion (Technosoft) específico del driver para intentar ver como se leen los datos del encoder. El primer paso para crear un nuevo proyecto es seleccionar el *axis number*. En segundo lugar indicamos el hardware. Se trata de un

sistema en bucle cerrado porque permite realimentación (*Closed Frame Drives*). Nuestro modelo de driver es IDM 680-8EI CAN_CANopen. El motor es rotatorio sin escobillas (*Brushless Rotary Motor*) y el encoder de tipo SSI.

Sin embargo, se resetea cuando intentamos darle a *Start* (*Project/Setup/TestConections*). Nos aparece el siguiente mensaje de error: *cannot find board with selected AxisID*. Acudimos al manual en busca de más información y verificamos que los interruptores están correctamente activados o desactivados (Ver *Ilustración 39*).



DIP Switch position					Axis ID
3	4	5	6	7	
ID – Bit4	ID – Bit3	ID – Bit2	ID – Bit1	ID – Bit0	
OFF	OFF	OFF	OFF	OFF	255
OFF	OFF	OFF	OFF	ON	1
OFF	OFF	OFF	ON	OFF	2
OFF	OFF	OFF	ON	ON	3
OFF	OFF	ON	OFF	OFF	4
OFF	OFF	ON	OFF	ON	5
OFF	OFF	ON	ON	OFF	6
OFF	OFF	ON	ON	ON	7
OFF	ON	OFF	OFF	OFF	8
OFF	ON	OFF	OFF	ON	9
OFF	ON	OFF	ON	OFF	10
OFF	ON	OFF	ON	ON	11
OFF	ON	ON	OFF	OFF	12
OFF	ON	ON	OFF	ON	13
OFF	ON	ON	ON	OFF	14
OFF	ON	ON	ON	ON	15
ON	OFF	OFF	OFF	OFF	16
ON	OFF	OFF	OFF	ON	17
ON	OFF	OFF	ON	OFF	18
ON	OFF	OFF	ON	ON	19
ON	OFF	ON	OFF	OFF	20
ON	OFF	ON	OFF	ON	21
ON	OFF	ON	ON	OFF	22
ON	OFF	ON	ON	ON	23
ON	ON	OFF	OFF	OFF	24
ON	ON	OFF	OFF	ON	25
ON	ON	OFF	ON	OFF	26
ON	ON	OFF	ON	ON	27
ON	ON	ON	OFF	OFF	28
ON	ON	ON	OFF	ON	29
ON	ON	ON	ON	OFF	30
ON	ON	ON	ON	ON	31

Ilustración 39: Configuración de la dirección a través de los interruptores del driver. Fuente: [26].

Pudiendo ser un fallo de potencia, elevamos la intensidad de alimentación como posible solución y volvemos a probar su funcionamiento. Como el error permanece, comprobamos con el osciloscopio las señales de reloj y de datos. Pero vemos que no hay cambios y el driver se vuelve a apagar. Por último, desconectamos el encoder y probamos por separado el driver. Tampoco funciona. Conclusión; el problema es del driver, de su firmware. No afecta el encoder.

Para poder seguir trabajando prescindimos del driver y emulamos su funcionamiento de forma manual. Generamos una señal de reloj con Simulink. Los pasos necesarios están descritos con detalle en el algoritmo del programa inicial. Se sitúa dentro del bloque creado por la UC3M para realizar una tarea continua (IDLE). (Ver *Ilustración 32 - C*).

Es muy importante a la hora de probar nuestro programa y cargarlo al microcontrolador saber en que lugar irá cada señal. Una característica de los bloques que hasta ahora no habíamos destacado es precisamente la elección del pin de la placa que vamos a usar. El dispositivo cuenta con dos grandes columnas de pares de puertos donde poder conectarse (Ver *Ilustración 40*).

Marcamos con un punto los que manejamos en el proyecto de control. El color naranja es del reloj que creamos en el primer planteamiento, pero después no usamos. Los de color rojo son los que se emplean en la nueva versión. También es importante distinguir las salidas que tiene la placa para las comunicaciones externas. La de bajo en el centro es para leer el microcontrolador. La situada en la parte superior nos sirve, tanto para transferir el programa al dispositivo, como para alimentar el dispositivo.

Ilustración 40: Sistema ARCP de control principal tipo STM32F4.

Dejamos de lado el driver y procedemos a leer directamente el dato dado por el encoder con el µC. Sin embargo, no podemos conectar la señal de reloj tal cual la hemos generado. Tenemos que convertirla en una señal balanceada para poder unir los pines del encoder, ya que por sus características de diseño se basa en una medida diferencial entre los polos +/-. Para eso utilizamos el transceiver MAX3077E que trabaja con 3,3V.

De su datasheet (Ver *Ilustración 41*) vemos como conectar los pines adecuadamente y obtenemos la señales apropiadas teniendo en cuenta que la patilla Y se refiere a la salida normal y la Z es la salida invertida. Observamos en el osciloscopio las señales polarizadas. Tienen simetría horizontal. Cuando una está a nivel alto la otra presenta el estado contrario.

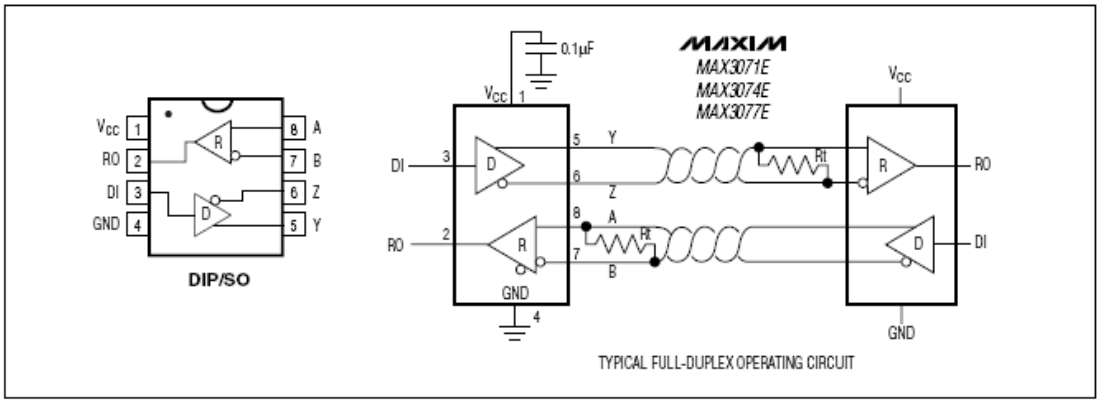


Ilustración 41: Esquema de conexiones para el integrado MAX3071E. Fuente: [27].

Para poder cargar el programa y visualizar las señales de reloj junto con el dato incluimos dos clabes para unir los puertos de entrada y salida correspondientes. En nuestra emulación se encuentran asignados como sigue: E/S del clk son PC10/PD0 y del dato PD1/PD2. Entonces visualimos la forma de ambas con el osciloscopio. Obtenemos una imagen parecida a la *Ilustración 31*, donde la parte inferior representa la información sobre la posición. Contando cuantos pulsos está a nivel alto y teniendo en cuenta el índice de bit que indique el reloj podemos extraer su valor númerico.

El inconveniente de este método es que no sabemos si el dato registrado es realmente la posición de la articulación, porque no hay opción a girar el sensor para ver si la lectura varía. Intentando solventar este hecho decidimos probar cuál debería ser el resultado de lectura final de un dato.

Para ello forzamos el número 5 utilizando los tres bits finales como explicamos en la tarea de tiempo continuo del primer algoritmo. Verificamos como el número de bit del dato coincide perfectamente con los pulsos de reloj del mismo índice. Además, podemos revisar si nuestras medidas de seguridad surgen efecto. Desconectamos la alimentación y tras un momento volvemos a enchufarlo. Vemos que el dato permanece constante, por lo que damos por bueno el sistema (Ver *Ilustración 42*).

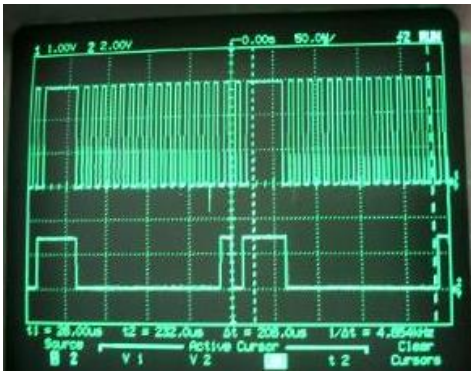


Ilustración 42:

Señales de reloj (superior) junto con un dato de valor 5 (inferior) en el osciloscopio.

En la nueva versión, en concreto, se corresponden con las siguientes representaciones de los fenómenos físicos. Los relojes (Clk) con subíndices que indican la referencia al sensor que perteneces son: PD0 - Clk_1, PD4 - Clk_2, PD6 - Clk_3, C12 - Clk_4. Designamos de igual forma los datos de posición que devuelven los encoders: PD1 - $data_1$, PD2 - $data_2$, PD3 - $data_3$, PD5 - $data_4$.

Establecemos las conexiones necesarias para poder confirmar que el programa definitivo funciona como hemos previsto. Distinguimos entre los microcontroladores. En el primero que tiene configurada la emulación del comportamiento de los encoders, solo es necesario enchufar la clavija inferior de la placa porque el objetivo es que podamos leer. Mientras que en la segunda placa queremos escribir nuestro algoritmo de control. Es indispensable unir la salida superior para cargar el programa.

Hay que unir tanto de las señales de reloj como de datos; salidas→entradas (Ver *Ilustración 43*). En el algoritmo hemos procurado asignar el mismo pin que los elegidos por Antonio Flores con el fin de hacer más intuitivo el conexionado. Sin embargo, en el caso del reloj del cuarto sensor, tuvimos que modificarlo para que la fuente IRQ del bloque correspondiente a su interrupción no coincidiese con el tercer encoder.

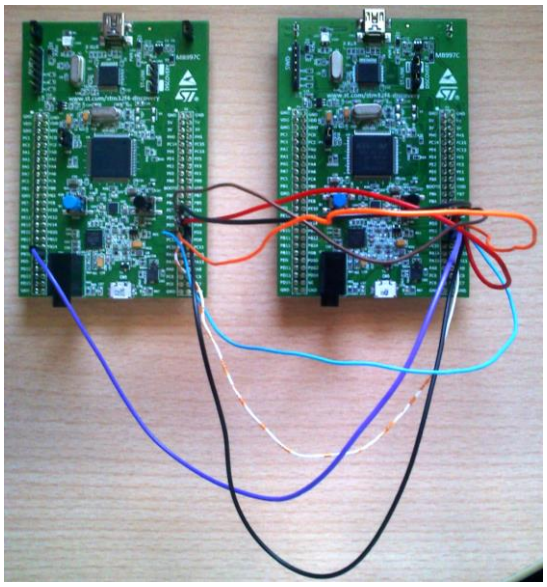


Ilustración 43: Conexiones de los microcontroladores para probar el programa definitivo.

Al acabar de establecer todos los enlaces pertinentes, manejamos los comandos del programa Simulink que nos permitirá obtener en la pantalla la información de control. Podemos comprobar el valor de posición de cada articulación. Comenzamos por revisar si hay algún error de sintaxis. Esto se hace a través del botón *Update Diagram* .

Cuando aparezca todo correcto, entonces enchufamos el µC y le damos a *Build Subsystem* . Nos aparecerá una ventana con diferentes etapas de carga (Ver *Ilustración 44*). Una vez que se hayan completado todas (se mostrarán en verde), podremos ver los resultados del sistema.

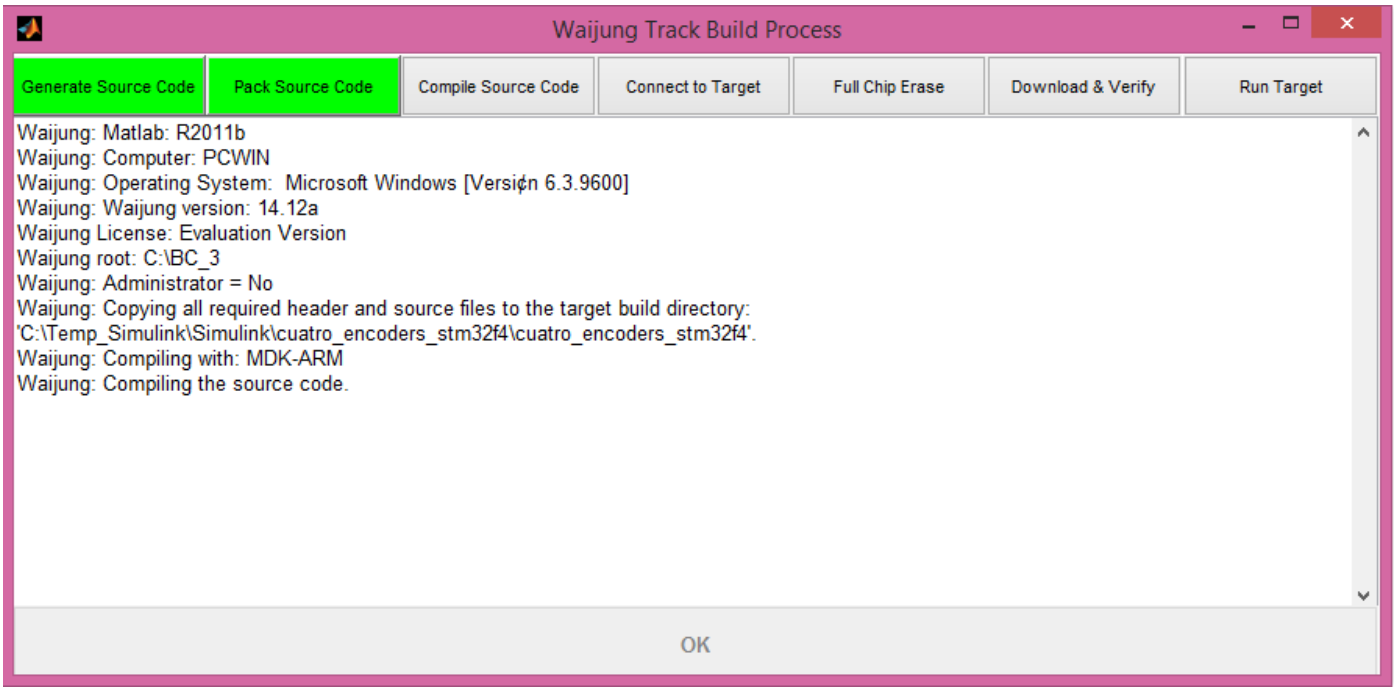


Ilustración 44: Etapas de carga del programa al microcontrolador

Otra forma de verlo es utilizar el osciloscopio. Situando los canales directamente en las salida de reloj y dato del MCU de emulación del sensor. Para analizar la cifra en formato decimal tenemos que convertir la secuencia binaria. A su vez, tenemos que incluir solo el nivel lógico que toma la señal en cada flanco de bajada del clk. (Ver Ilustración 45). Siguiendo este procedimiento, obtenemos como resultado de posición de los encoders las siguientes cantidades: 89, 35, 76 y 108.

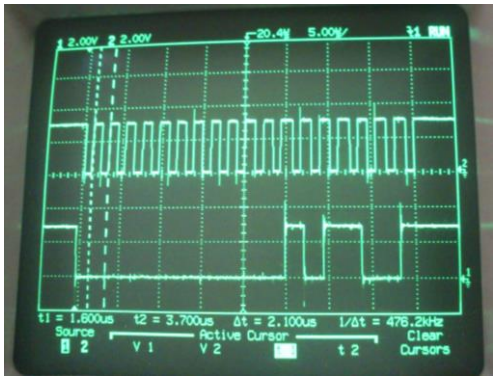


Ilustración 45: Lectura del encoder 1.

3. CONCLUSIONES

Después de explicar detalladamente el proyecto, considero que se han cumplido todos los objetivos propuestos desde un principio. Logramos utilizar los elementos del robot de forma eficiente para poder integrarlo de forma adecuada en nuestra aplicación donde se requieren respuestas rápidas y controladas.

Para conseguirlo ha sido necesario la experimentación, ya que desarrollar ideas teóricas por si solas no tiene sentido si se trata de forma independiente al sistema real que afectan. Podemos resaltar las diferencias que suponen estos diferentes métodos de trabajo en nuestro caso (Ver *Ilustración 46*).

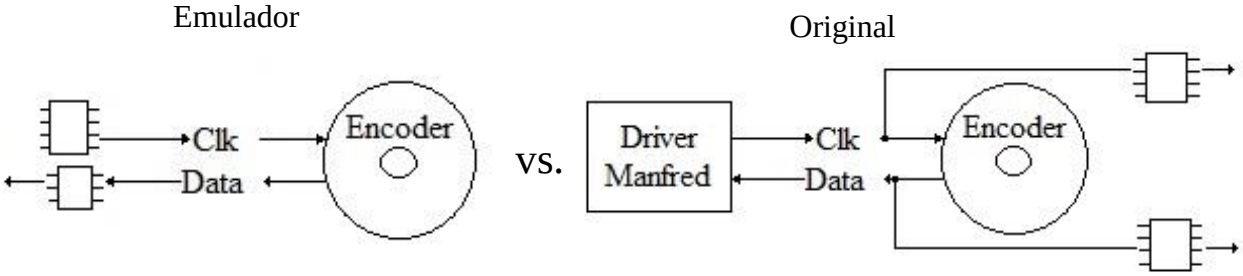


Ilustración 46: Diferencias entre el sistema real y la emulación.

Comparando ambas imágenes destacamos los factores principales que las distinguen. Por un lado, el número de componentes es mucho menor en la emulación, lo que supondrá ahorros en coste de los recursos materiales. En contraposición, desde el punto de vista del software es más sencillo el original porque la señal del reloj es dato.

Este proyecto ha sido además indispensable para satisfacer los objetivos generales de la carrera de Ingeniería en Tecnologías Industriales. Gracias a todo lo aprendido en estos meses considero se han mejorado los recursos curriculares del estudiante, tácticas a la hora de poner en práctica los conocimientos y técnicas aprendidas desde el comienzo de los estudios de grado. Quiero destacar la gran utilidad del lenguaje gráfico con el que hasta este momento no había tenido la oportunidad de trabajar.

4. PRESUPUESTO

Plantearse dar respuesta a un problema y afrontar el trabajo hasta conseguirlo implica también efectos económicos. No se puede dejar de lado el coste que supone. Además, incluye factores tanto materiales como humanos. A continuación resumimos en una tabla una lista del coste de los elementos influyentes en nuestro proyecto.

Detalle de presupuesto	Coste (€)
Componentes hardware	
Driver de los motores	350€ / unidad
Placa Microcontrolador STM32F4	15€ / unidad
Placa Microcontrolador STM32F103 (RS-232 a USB)	20 €
Transceiver MAX3077E	0.5€ / unidad
Componentes software	Adquirida por la universidad
Licencia del programa Matlab/Simulink	
Horas de ingeniería	30€/h * 100 h = 30000 €
Horas de documentación	18€/h * 15 h = 270 €
Gastos generales	10% del total
Consumo eléctrico	
Alimentación	
Desplazamientos	
Impuestos	21% del total
IVA en ingeniería	

Es importante resaltar que el principal gasto es debido al trabajo personal del ingeniero. Esto es compatible con las tecnologías de sistemas ARCP, ya que los componentes hardware no necesitan mucha inversión monetaria.

5. RECURSOS BIBLIOGRÁFICOS

[1] Blog de Tecnología e informática. Myprofeciencias. *Historia y evolución de la tecnología*. Publicado el 6 de febrero de 2011 en https://myprofetecnologia.wordpress.com/2011/02/06/historia-y-evolucin-de-la-tecnologia/

[2] Ruiz Rojas, Paula Andrea. *Mecatrónica. Revolución del siglo XXI.* Artículo extraído de la revista Metal Actual, en la sección de tecnología. Pp. 46-53.

[3] Olivares S., Manuel (2014). *Instrumentalización y control de sistemas mecatrónicos.* Publicado en octubre, en Electro Industria. Disponible en http://www.emb.cl/electroindustria/articulo.mvc?xid=2394&xit=instrumentacion-y-control-de-sistemas-mecatronicos

[4] Mecatrónica (2012). *Sistema mecatrónico: sensores, control y actuadores.* Publicado el 2 de noviembre en línea, en https://jornadamecatronico.wordpress.com/2012/11/12/sistema-mecatronico-sensores-control-y-actuadores/

[5] Torres, Samuel. *Integración de tecnologías de automatización.* Disponible en línea en http://www.aie.cl/files/file/comites/ca/abc/Integracion_de_Tecnologias.pdf

[6] Coronado Cabrera, Emiro; Correa, Manuel (2009). *Objetos sensores y actuadores en la integración de sistemas heterogéneos.* Artículo publicado en ACADEMIA, Trujillo, Venezuela. Recibido el 8/09/09. ISSN: 1690-3226. Julio-Diciembre, volumen III (16). Pp. 82-86.

[7] Fundación OPTI- Observatorio de Prospectiva Tecnológica Industrial (Marzo 2010). *Oportunidades tecnológicas e industriales para el desarrollo de la economía española.* Resumen 142 publicado en la página oficial de la fundación, en www.opti.org

[8] G. Dean, Alexander; M. Conrad, James (2012). *Creating Fast, Responsive and Energy-Efficient. Embedded Systems using the Renesas RL78 Microcontroller*. Extraído de la plataforma Renesas Books en http://am.renesas.com/support/books/ ISBN: 978-1-935772-98-9.

[9] Echávarri Otero, Javier; Carbone, Giuseppe; Ceccarelli, Marco y Muñoz Sanz, Jose Luis (2007). *Criterios para la seguridad en el uso de robots.* Documento redactado a partir de las conferencia del 8º Congreso Iberoamericano de Ingeniería Mecánica, que se realizó en Cusco del 23 al 25 de octubre. Publicado en Academia Edu, disponible en http://www.academia.edu/2658695/CRITERIOS_PARA_LA_SEGURIDAD_EN_EL_USO_DE_ROBOTS

[10] IEC- International Electrotechnical Commision (2015). *Funtional safety and IEC 61508.* Disponible en http://www.iec.ch/functionalsafety/

[11] *Functional safety and IEC 61508: A basic guide* (Mayo 2004). Disponible en http://www.ida.liu.se/~simna73/teaching/SCRTS/IEC61508_Guide.pdf

[12] Flores Caballero, Antonio. (Diciembre 2014). *Sistema avanzado de prototipado para control en exoesqueletos y dispositivos mecatrónicos.* Tesis doctoral ubicada en el archivo online de la universidad Carlos III de Madrid. Disponible en http://e-archivo.uc3m.es/handle/10016/20665

[13] LBA Industrial Mining. *Qué es un encoder? Cómo funciona? Tipos de encoder que existen.* Publicado en el blog de la Compañía S. de R.L. de C.V. Disponible en http://www.lbaindustrial.com.mx/que-es-un-encoder/

[14] Torres Rodriguez, Sergio I. *Universal serial bus.* Disponible en http://www.iuma.ulpgc.es/~avega/int_equipos/trab9899/usb_1/index.html

[15] Definición de Transceiver. Publicado en Master Magazine; disponible en http://www.mastermagazine.info/termino/6937.php

[16] Guerrero Martínez, Juan F.; Francés Villora, José V. (Curso 2010/11). *Computadoras RISC.* Disponible en http://ocw.uv.es/ingenieria-y-arquitectura/sistemas-electronicos-para-el-tratamiento-de-la-informacion/seti_materiales/seti6_ocw.pdf

[17] ARM Power. *Introducción práctica a los microcontroladores ARM Cortex-M.* Publicado en el blog http://armpower.blogs.upv.es/materiales-arm-cortex-m-iniciacion-2/

[18] García Valderas, Mario; Patón Álvarez, Susana (Curso 2014/15). *Sistemas Electrónicos Digitales.* Apuntes de la asignatura impartida en la Universidad Carlos III de Madrid, Campus de Leganés. Los temas consultados son 'Arquitectura y estructura de computadores', 'Programación en C para microcontroladores Aplicación a RL78/G14' y 'Puertos de Entrada/Salida y Conceptos básicos de interrupciones y temporización'.

[19] Flores-Caballero, Antonio; Copaci, Dorin; Blanco, María Dolores; Moreno, Luis; Herrán, Jaime [et al] (2014). *Innovative Pressure Sensor Platform and Its Integration with an End-User Application.* Publicado en Sensors el 11 de junio, 14, 10273-10291; doi: 10.3390/s140610273.

[20] Mathworks. *About Mathworks: Founders.* Disponible en http://es.mathworks.com/company/aboutus/founders/

[21] Jimenez, Raúl. Trabajo sobre sistemas empotrados. *Tema 1, Introducción.* Universidad de Huelva. Disponible en http://www.uhu.es/raul.jimenez/EMPOTRADO/introduccion.pdf

[22] Empresa Keil, tolos by ARM. Entrada de la guía del usuario: *Idle task. Disponible en* http://www.keil.com/support/man/docs/rlarm/rlarm_ar_cfgidle.htm

[23] Manuales de la empresa Netzer. "DS-70 Absolute position, rotary Electric Encoder™" (Data Sheet, V 2.0a ,April 2014) y "DS-58[20] Absolute position, rotary Electric Encoder™" (Data Sheet , V 2.0c, MAR 2014). Disponibles en http://www.netzerprecision.com/products.asp

[24] Manual de la empresa STMicroelectronics. *Datasheet STM32F405xx y STM32F407xx*. DocID022152 Rev 4. Junio 2013. Disponible en http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/LN1848/PF252419

[25] Definición de Síncrono. Publicado en Master Magazine; disponible en http://www.mastermagazine.info/termino/6678.php

[26] Manual de la empresa Technosoft. *IDM680, Intelligent Servo Drive. Technical Reference* (2011).

[27] Manual de la empresa Maxim Integrated Products. *MAX3070E-MAX3079E.* Con referencia *19-2668; Rev 1; 1/03*. Disponible en http://www.maximintegrated.com/en/products/interface/transceivers/MAX3077E.html

[28] Foro Diydrones. The leading Community for Personal UAVs. Respuestas a la pregunta escrita por Nigel el 22 de mayo de 2011. *Baud Rates - Why don't we use 115200 all the time?* Disponible en http://diydrones.com/forum/topics/baud-rates-why-dont-we-use

[29] Publicaciones de Flores Caballero, Antonio. Dr. Ingeniería Eléctrica, Electrónica y Automática - PhD Degree. Disponibles en http://www.researchgate.net/profile/Antonio_Flores-Caballero/publications

[30] Tutorial de la compañía Microchip. La arquitectura Harvard vs. Von Neumann. El set de instrucciones RISC. Procesamiento *pipeline*. Disponible en http://www.puntoflotante.net/TUTORIAL%20ARQUITECTURA%20HARVARD.htm

[31] Docsetools. *Lenguajes de programación síncrona*. Disponible en http://docsetools.com/articulos-educativos/article_10441.html

[32] Video de Gordon, Ray de la empresa Mathworks. *Introduction to Simulink.* Publicado el día 17 de abril de 2014 en http://es.mathworks.com/videos/introduction-to-simulink-81623.html

[33] Video de Mahapatra, Saurabh de la empresa Mathworks. *Modeling Discrete-Time Systems with MATLAB and Simulink.* Publicado el día 12 de diciembre de 2013 en http://es.mathworks.com/videos/modeling-discrete-time-systems-with-matlab-and-simulink-87583.html

[34] Video de Ananthan, Arvind de la empresa Mathworks. *Developing IEC 62304 Compliant Medical Device Software Using Model-Based Design*. Publicado el día 22 de octubre de 2013 en http://es.mathworks.com/videos/developing-iec-62304-compliant-medical-device-software-using-model-based-design-86469.html

[35] Video de Erkkinen, Tom de la empresa Mathworks. *ARM Cortex-M Optimized Code from MATLAB and Simulink.* Publicado el día 26 de septiembre de 2013en http://es.mathworks.com/videos/arm-cortex-m-optimized-code-from-matlab-and-simulink-82000.html

[36] Publicación de SalesEng el día 25 de julio de 2012 en la web Opengear Help Desk. *RS232, RS422 & RS485 standard DB connector pinout*. Disponible en https://opengear.zendesk.com/entries/21745261-RS232-RS422-RS485-standard-DB-connector-pinout

[37] Empresa IFM electronics. *Sensores para el control de movimiento. Encoders. Descripción del sistema.* Disponible en http://www.ifm.com/ifmes/web/pinfo015_010_040.htm

[38] Empresa National Instruments. *A Quick Comparison of RS-232, RS-422, and RS-485 Serial Communication Interfaces.* Publiación del día 28 de septiembre de 2014. Disponible en http://digital.ni.com/public.nsf/allkb/2CABB3FD5CAF2F8686256F1D005AD0CD

[39] Diapositivas del master de Rodriguez, Oscar. Asignatura de robótica. (06-May-2011) *Fundamentos de la robótica*. Disponible en http://www.intechap.us/files/Hablando_el_Lenguaje_de_la_Robotica_Ver4.pdf

[40] Entrena, Luis; López, Celia; García, Mario y San Millán, Enrique. Profesores de la Universidad Carlos III de Madrid. *Biestables*. Disponible en http://ocw.uc3m.es/tecnologia-electronica/electronica-digital/espanol_pdf/tema05.biestables

[41] Paperblog. Brenchat Antolin, Oscar. *Encoder Incremental*. Publicado el día 16 de febrero de 2014 en http://es.paperblog.com/encoder-incremental-2430448/

[42] Foro técnico de fibra óptica: fibraopticahoy. *Encoder de fibra óptica*. Publicado el 23 de mayo de 2012 en http://www.fibraopticahoy.com/encoder-de-fibra-optica/

[43] Scielo. Artículo Información *Tecnológica.* Vol. 17 N°6-2006, págs.: 7-12. Disponible en http://www.scielo.cl/scielo.php?pid=S0718-07642006000600003&script=sci_arttext

[44] Foros de Electrónica. Comunidad Internacional de Electrónicos. *Encoders: Información Técnica*. (2005) Disponible en http://www.forosdeelectronica.com/f16/encoders-informacion-tecnica-25/

6. ANEXOS

-Diccionario de abreviaturas

ARCP	Advanced Rapid Control Prototyping
ARM	Advanced RISC Machines
ASM	Assembler
CAN	Controller Area Network
CISC	Complex Instruction Set Computer
Clk	Señal de reloj (*clock*)
COM	Puertos destinados a comunicaciones en un sistema embebido
CPU	Central Processing Unit
CSI	Clocked Serial Interface
CU	Control Unit
D	Tipo de biestable síncrono activo por nivel. Si está habilitado (1 lógico) la salida toma el valor de la entrada D. Si no, mantiene su valor.
DAC	Digital Analogic Conversion
DMA	Direct Memory Access
DSP	Digital Signal Processor
E/S	Entradas/Salidas
EXTI	External interrupts
GND	Referencia de masa o tierra (*ground*)
IC	Integrated Circuit
ID	International Designator
IEC	International Electrotechnical Commision
IIC	Inter-Integrated Circuit
IRQ	Interrupt ReQuest
ISO	International Organization for Standardization
ISR	Interrupt Service Rutine
µC	Microcontolador
MCU	MicroController Unit
MUX	Multiplexor
PCB	Printed Circuit Board
PID	Proportional-Integral-Derivative controller
PWM	Pulse-Width Modulation
RAI	Rutina de Atención a la Interrupción

RCP	Rapid Control Prototyping
RISC	Reduced Instruction Set Computer
RPM	Revoluciones Por Minuto
RS	Recommended Standard
SIL	Safety Integrity Level
SPI	Serial Peripheral Interface
SSI	Synchronous Serial Interface
TIC	Tecnologías de la Información y Comunicación
TIMx	Puertos específicos para los temporizadores del microcontrolador
UART	Universal Asynchronous Receiver-Transmitter
UC3M	Universidad Carlos Tercero de Madrid
UP	Unidad de Procesamiento de datos
USB	Universal Serial Bus
VCP	Virtual COM Port